Mining Frequent Patterns in Uncertain and Relational Data Streams using the Landmark Windows

Fatemeh Abdi[1], Ali Asghar Safaei[2]

Received (2015-09-09)
Accepted (2015-12-14)

***Abstract* - Todays, in many modern applications, we search for frequent and repeating patterns in the analyzed data sets. In this search, we look for patterns that frequently appear in data set and mark them as frequent patterns to enable users to make decisions based on these discoveries. Most algorithms presented in the context of data stream mining and frequent pattern detection, work either on uncertain data, or use the sliding window model to assess data streams. Sliding window model uses a fixed-size window to only maintain the most recently inserted data and ignores all previous data (or those that are out of its window). Many real-world applications however require maintaining all inserted or obtained data. Therefore, the question arises that whether other window models can be used to find frequent patterns in dynamic streams of uncertain data.**

In this paper, we used landmark window model and time-fading model to answer that question. The method presented in the form of proposed algorithm, which uses the idea of landmark window model to find frequent patterns in the relational and uncertain data streams, shows a better performance in finding functional dependencies than other methods in this field. Another advantage of this method compared with other methods is that it shows tuples that do not follow a single dependency. This feature can be used to detect inconsistent data in a data set.

***Keywords* - data stream, landmark window, data dependency, sliding window, time-fading window, relational and uncertain data streams**

1- Nima Institute, Mahmoodabad, Mazandaran, Iran. (fasaabdi@nima.ac.ir)
2- Department of Biomedical Informatics, Faculty of Medical Sciences, Tarbiat Modares University, Tehran Iran. (aa.safaei@modares.ac.ir)

1. Introduction

Frequent pattern mining is an important field of research related to database and data mining. This process finds the patterns that appear frequently in data set and marks them as frequent patterns. This method in used in many everyday applications, such as data set related to supermarket sales, data set related to the weather reports and environmental data sets. There have been many studies in the field of frequent pattern mining, and most of them have been focused on exact and precise data and on cases where users have accurate information about the items in the data set. But in most everyday (real world) situations, users do not have precise information about the presence or absence of items in the data set (data sets related to quantum physics, thermal sensors, and environmental monitoring). This type of data is called uncertain data. In uncertain data, the presence or absence of an item is represented with a probability value (also called uncertain value). Furthermore, Data sets used in the real world are not static and are often in the form of rapid stream of new data. Mining this type of dynamic data set, is called data stream mining.

Apriori algorithm [1] was presented by Agrawal and Srikant in 1994 for mining precise static data sets. Any pattern related to data set has a support value that describes the number of pattern repetition. A pattern (or item set) is considered frequent only when support value of that pattern is greater than or equal to the minimum support threshold set by the user. Apriori uses a "bottom up" approach to test the data and determine frequent patterns in given data sets. But in 2000 Han et al [3] introduced FP-growth algorithm which avoids the process of candidate generation. This algorithm is composed of two processes: 1-constructing FP tree and 2- recursively growing frequent patterns. FP Tree is an extended Prefix Tree which includes the contents of data sets. Algorithm needs to examine and re-examine the data set to form FP Tree. In First examination, algorithm looks for single patterns. It removes all unique patterns and then arranges all frequent single patterns in order of their support value. FP tree potential value decreases similarly. Then, algorithm examines the data set for the second time to form the FP tree. After the construction of this tree, data set examination is no longer needed, because FP Tree includes all information provided by data set. Then all frequent patterns can be obtained by a recursive growth process in each branch of this tree.

In 2004, Giannella et al proposed FP-Streaming algorithm [3] which is an algorithm for mining precise data streams. The first stage of FP-streaming is to call FP-growth with a preMinsup to mine the current data stream. At first, FP-growth algorithm finds all frequent patterns; the next step of FP-streaming algorithm is storing and maintaining these patterns in another tree structure which is called FP-stream. In FP-streaming, each path represents a frequent pattern. Each node in FP-streaming includes a sliding window table which contains multiple support values for each batch of transactions. It should be noted that FP-streaming is a data stream mining algorithm for precise data. The real challenge is in processing uncertain data streams. UF-streaming, which is proposed by Leung and Hao, mines frequent patterns from uncertain data streams using a fixed-size sliding window of w frequent steams. At first, UF streaming algorithm calls UF-growth to find frequent patterns from the current batch of transactions in the streams (using minimum support threshold). A pattern is frequent when its support value is greater than minimum support threshold. Then UF-streaming stores the frequent patterns and their support values in a tree structure, so that in each node, X stores a list of w support values. When a new stream arrives, the window slides and support values move to mine frequent patterns and their expected support values from the newest stream imported into the window, and those streams which are found to be oldest in the window are deleted. This process is repeated for each batch in the stream. The support value of each frequent pattern X will be calculated by summing all w support values (one for each batch in the sliding window). expSup(X, Bi) represents the support value of X in batch Bi. Then, at the moment of T, the support value of X in the current sliding window which contains w batches of uncertain data in the form of $B_{T-w+1},...,B_T$ is calculated as below:

$$\exp Sup(X, \bigcup_{i=T-w+1}^{T} B_i) = \sum_{i=T-w+1}^{T} \exp Sup(X, B_i)$$

• sliding window model

The sliding window model needs a user to define the size of the window and the extent of window slide over certain time periods [15]. As can be seen in the figure, vectors represent the windows. At first, window W covers the first data segment. When the next time period arrives, the sliding window grows to size 2. Then, in the third time period, the sliding window grows to size 3, which is the maximum capacity of the sliding window in this model. In the next time period window slides one segment to the right side and covers data segment 2 to data segment 4, and then deletes the data segment in the right end which is the oldest data segment to maintain the size of the window. The process is then repeated.

• Landmark window model

The landmark window model is not a fixed-size window technique [16, 34]. It starts from a given point and ends at the current time and increases the size of the window. It should be noted that the starting point is fixed, but over time the end point shifts. As can be seen in the figure, the landmark window model starts from the leftmost data segment. The window consists of data segment 1. In the next time period, the end-point of the landmark window moves to the next data segment, but the start-point remains fixed. Over time, the size of the landmark window increases and stores every data segment (figure 1).

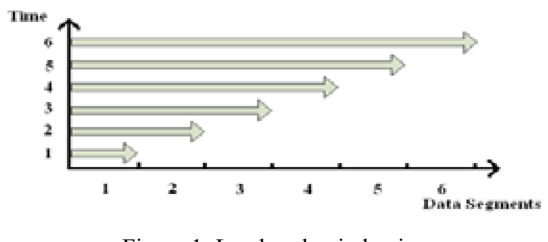


Figure 1: Landmark windowing

• Conditional Restrictions

Here, we review the important conditions or the upper and lower bounds of attribute domain. We select the minimum and maximum values for all attributes in all relationships with the corresponding SQL commands. SQL commands use a normal order on numbers for numerical attributes and use the lexicographic order on the character set for attributes of a symbolic type. It is possible to compute two values in one query, so the overall computation cost is O(n*m) where n is the number of attributes in all tables and m is the maximum number of tuples in the table.

• Functional dependencies

The functional dependencies are vital and necessary for the design of relational databases, so studying these dependencies and their application is of significant importance. Functional dependency has been thoroughly studied for decades. Functional dependency is the relation between attributes of a relationship: Functional dependency expresses the value of an attribute by using the value of some other unique attributes. Classic functional dependencies are mostly used for designing relational databases, normalizing relations, and to avoid data redundancy and key anomalies. These dependencies have no flexibility against exceptions and potential noise in data. But approximate dependencies are another type of dependencies which do not necessarily hold for the entire data set, and thus are flexible against noise and exceptions.

The rest of the paper is organized as follows: the proposed method which uses Landmark windowing for mining frequent patterns in relational and uncertain data streams is presented in section 2. Theoretical analysis and experimental evaluation of the proposed method are presented in section 3 and section 4, respectively. Some of the related works are discussed in section 5. Finally, the paper is concluded in section 6.

2. The proposed method

The landmark window model is not a fixed-size window technique. It stores all data segments from the start point, which is a given time, up to the end point, which is the current time. This model considers all data to be of the same importance, so it does not discard any data. In this model, the size of the window is also considered to be Incremental. Our proposed algorithm operates as follows. Assume a stream of uncertain data and a minimal threshold defined by user (Minsup). The proposed LUF-Streaming algorithm uses landmark window model in an uncertain data environment to mine frequent patterns of data stream (i.e., expected support value is greater than or equal to minimum support threshold Minsup). The algorithm consists of three main models: batch mining model, stream mining model, and pruning model.

• Cg Batch mining model

This model finds the frequent patterns in every batch of the stream by calling the UF-growth algorithm. In a stream, Data do not necessarily have uniform distribution; so a unique pattern may be repeated later. Here, instead of using the UF-growth algorithm with a support threshold of minsup, model uses UF-growth algorithms with a lower support threshold of preMinsup and thus avoids patterns being discarded. Therefore, model finds each sub-frequent pattern *X* (i.e. *X* with an expected support value greater than or equal to preMinsup) in each batch of uncertain data in the stream. The support value of *X* in the batch ***B**$_j$* can be calculated by summing (most transactions in ***B**$_j$*) the product (probabilities of independent items apearing in the pattern *X*) which is expressed by the following equation [36]:

$$\exp Sup(X, B_j) = \sum_{i=1, t_i \in B_j}^{w} \left(\prod_{x \in X} P(x, t_i)\right)$$

where w is the number of batches in the stream.

• Stream mining model

This model stores and maintains the sub frequent patterns, discovered in the batches of the stream, in a tree structure called LUF-Streaming where each node in the tree represents a pattern and contains information about the support value in the stream (from the start of landmark up to the present).

• Pruning model

In the end, when the user requests the final results, this model prunes the sub frequent patterns that are definitely not frequent. Note that to avoid rapid pattern pruning, batch mining model finds the sub frequent patterns that have a support value greater than or equal to preMinsup that may be less than minsup. These sub frequent patterns will be maintained in LUF-Streaming and their expected support value will be updated by stream mining model. When user requests the final list of frequent patterns, pruning model removes the patterns that have a support value greater than or equal to preMinsup and less than minsup.

3. Analyses of overheads

In the functional dependency detection stage, Assume that | R | = n is the number of attributes in relational schema R, |r| = m is the number of attributes in relation r (with a structure of R), |fr|= f is the number of evaluated dependencies, |LHS| is the maximum number of attributes in the left hand side of dependencies, and |D| = d is the average number of discrete values in the attribute domain(i.e. the number of members of MB-Set). In the presented algorithm, calculation of MB-Set is of the order of O (n.m) and the construction of set M for a dependency is of the order of O(d|LHS|). Since for each member of M, d logical AND operations must be performed, the complexity of the F set calculation will be of the order of O(d|LHS|+1) and overall complexity of the algorithm will be O(n.m + f.d|LHS|+1). The best condition occurs when the development of left hand side of dependency is not needed, or in other words, when the accuracy of all dependencies that contains 1 attributes on the left hand side are acceptable. In this condition, |LHS| = 1, and f is at its lowest value i.e. n2. So in the best condition, the overall complexity of the algorithm is O(n.m + n2.d2).

The worst condition occurs when there is no regularity among data and when few attributes on the left hand side do not give us any interesting dependency. In this condition, the left hand side of all dependencies should be developed to the point that no other attribute remain to be added to the left hand side. So | LHS | will be equal to n- 1 and the total number of dependencies that will be tested will include the whole search space (i.e. 2n dependencies). But in practice, this condition is very rare especially in the case of real data. In the following, we will introduce five algorithms for finding frequent patterns from streams of uncertain data by using the landmark window model and the time-fading model. In this section, $|fp_i|$ represents the number of frequent patterns mined from the batch. We assess the proposed algorithm from two aspects: memory usage and runtime.

• Memory usage

When using landmark window model, LUF-Streaming algorithm needs $|\cup_i fp_i|$ expSup values (to be stored in the LUF-stream to store frequent pattern information.

• Runtime overhead

for the landmark window model, the LUF-streaming first needs to visit/update every node in the LUF-stream tree for each data batch. However, we can increase the performance of the method in a way that it only visits/updates those tree nodes that correspond to frequent patterns in the next data stream. Except that they need a lesser runtime compared with time-fading model and this is because of simpler calculations. Simplification of Equation also simplifies the calculation of support values when α = 1 (for the landmark model) because the terms $[\exp Sup(X,\bigcup_{i=1}^{T-1} B_i) * \alpha]$ and $[\exp Sup(X,\bigcup_{i=1}^{LV} B_i) * \alpha^{T-LV}]$ can be simplified to $(X,\bigcup_{i=1}^{LV} B_i)$ and $(X,\bigcup_{i=1}^{T-1} B_i)$.

4. Experimental evaluation

In this section, we evaluate the proposed method and the performance of algorithms by using two following real data sets acquired from UCI data repository.

1)Nursery data set

2)Mushroom data set [37].

Overall statistics regarding these data sets is presented in table 1.

TABLE 1
DESCRIPTION OF USED DATA SETS

Data set	Number of attributes	number of tuples
Nursery	9	12960
Mushroom	22	8124

Presented data sets also have some attributes with continuous numerical values. Before the start of main operation, the values of these attributes were converted to discrete values by partitioning their domains into 5 equal parts. To equalize the tests conditions for two algorithms, we set the accuracy threshold to 1.0. Runtime performance of AD-Miner algorithm (which is more efficient than other dependency mining methods) is obtained for data sets and the results are shown in table 2.

TABLE 2
DURATION OF DATA MINING PROCESS FOR USED DATASETS

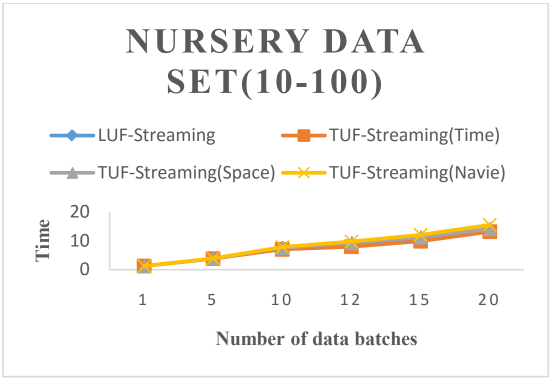
Datasets	AD_Miner (s)
Nursery	82
Mushroom	54

AD-Miner algorithm is able to mine dependencies with any degree of accuracy.

These data sets are pretty well-known in the field of data mining. For Nursery data set, we prepare two sets of data; (50-60) and (10-100). These ranges represent the occurrence probability in the data set. For example, (50-60) in the data set means that all item sets (patterns) will have support values in the range between 0.5 and 0.6. Similarly, (10-100) in the data set means that all patterns will have support values in the range between 0.1 and 1.0. Nursery data set consists of 12960 streams. We divide these streams into 20 separate data batches (each batch consists of 648 streams). For Mushroom data set, there are a total of 8124 streams. We also divide these streams into 20 data batches (On average, each batch contains 406 streams or transactions). For each of these data sets, we examine the proposed method in terms of different aspects: (1) The effect of number of batches on the runtime, and (2) the effect of threshold minsup set by the user on the runtime. Then, we select the algorithms that show the best performance to compare them with another algorithm for mining the streams of uncertain data, which is UF-streaming algorithm.

•Evaluation of different numbers of data batches

Experiment 1. In this experimental evaluation, we compared the runtime of algorithms by running different numbers of batches on Nursery data set. In all steps of algorithm, the support threshold minsup was assumed as 1.2 (figure 2).



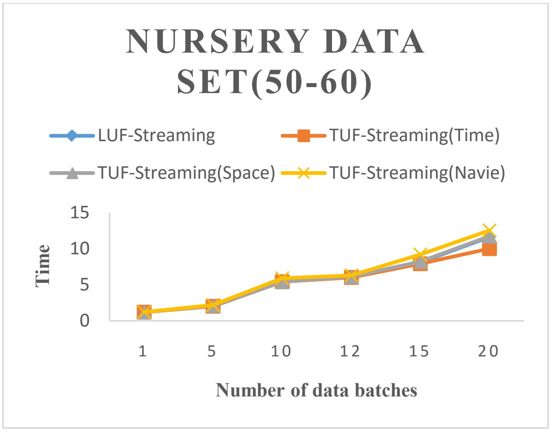


Figure 2: evaluation of different numbers of data batches on Nursery data set (experiment 1)

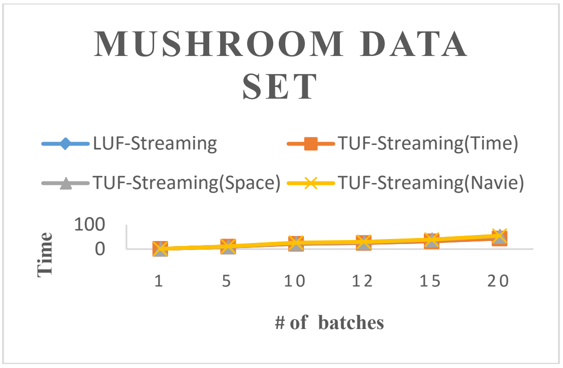


Figure 3: evaluation of different numbers of data batches on Mushroom data set (experiment 2)

Experiment 2. Figure 3 shows the result of running algorithms on mushroom data set. In this section, we again compare the algorithms runtimes for different numbers of batches. We again assume the support threshold as 1.2 and the α value for all TUF-Streaming algorithms as 1. The difference between the performances of algorithms is much clearer in the Mushroom data set compared with Nursery data set. The reason behind this difference is that the result obtained by mining mushroom data set contains more frequent patterns compared with Nursery data set, which leads to a larger TUF-stream tree structure (or

LUF-Streaming structure). So the performance of tree updating process becomes much better. Overall, the TUF-Streaming (simple) has the longest runtime and LUF-Streaming and TUF-Streaming (Time) have the shortest runtime.

Experiment 3. Both LUF-Streaming and TUF-Streaming (Time) show the best performance in terms of runtime. We use the same method also used in previous experiments to compare these two algorithms with the existing algorithm. At first, we run all algorithms for Nursery data set (Fig. 3) and Mushroom data set (Fig. 4) by an increasing numbers of batches. A possible argument may be that different window models are used for these algorithms, so the results may be different. The result of this experiment shows the strength of proposed algorithm in using data streams. This method also provides readers with some idea regarding the runtime with respect to importance (magnitude) when making a comparison between algorithms.

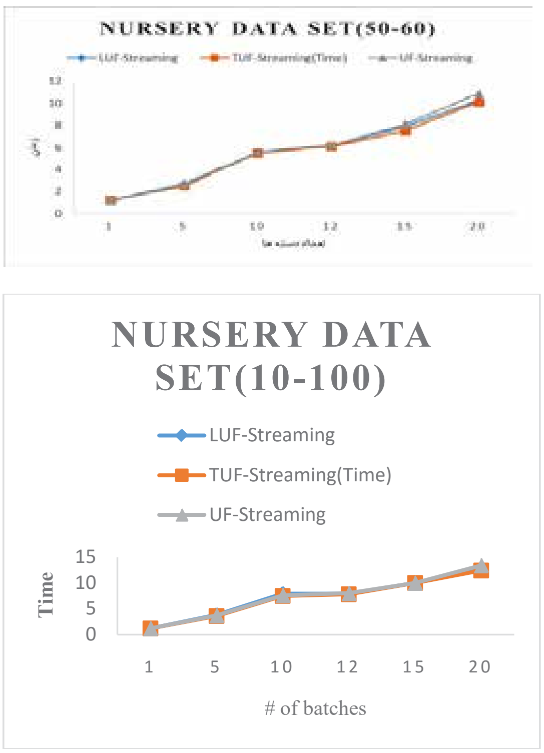


Figure 4: evaluation of different numbers of data batches on Nursery data set (experiment 3)

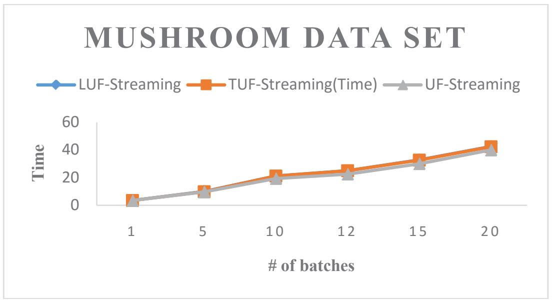


Figure 5: evaluation of different numbers of data batches on Mushroom data set (experiment 3)

• Evaluation of different thresholds

Experiment 4. In this experiment, we assess all proposed algorithms with different thresholds on both Nursery data set (figure 6) and Mushroom data set (figure 7). In both figures, x-axis represents the threshold, and y-axis represents the runtime. We assume the number of data batches in each run to be 10. To make sure that our assessment is correct, we assume the value of α for all TUF-Streaming algorithms as 1. The results show that LUF-Streaming and TUF-Streaming (Time) have the shortest runtimes among the tested algorithms.

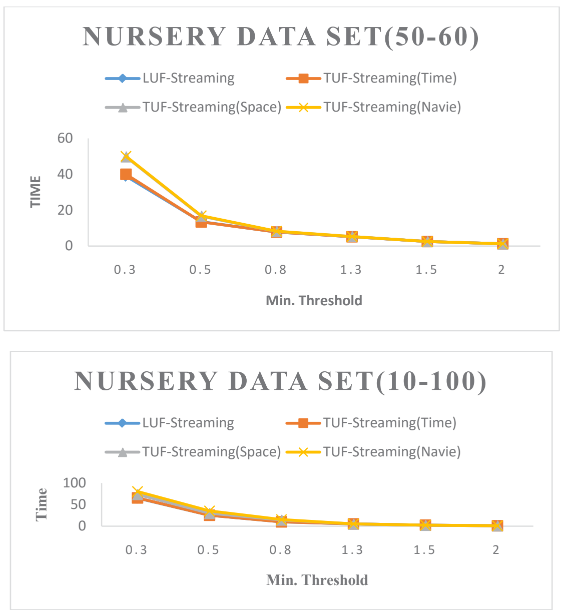


Figure 6: evaluation of different thresholds on Nursery data set (experiment 4)

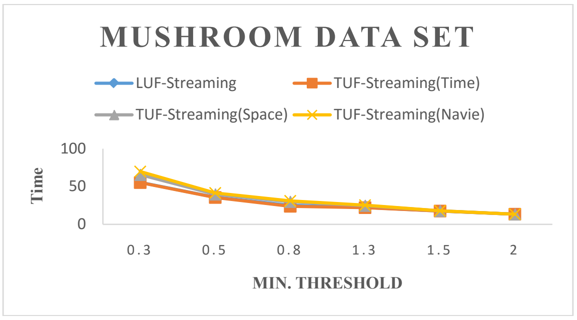


Figure 7: evaluation of different thresholds on Mushroom data set (experiment 4)

Experiment 5. This experiment compares TUF-Steaming (Time) and LUF-Streaming and UF-Streaming running on the Nursery data set (figure 8) and Mushroom data set (figure 9) with different thresholds. According to Experiment 3, we know that the number of frequent patterns returned by UF-streaming is different from the number of frequent patterns returned by proposed algorithms. This is because the UF-Streaming uses sliding window to remove data. However, we see that our algorithms are only slightly slower than UF-streaming.

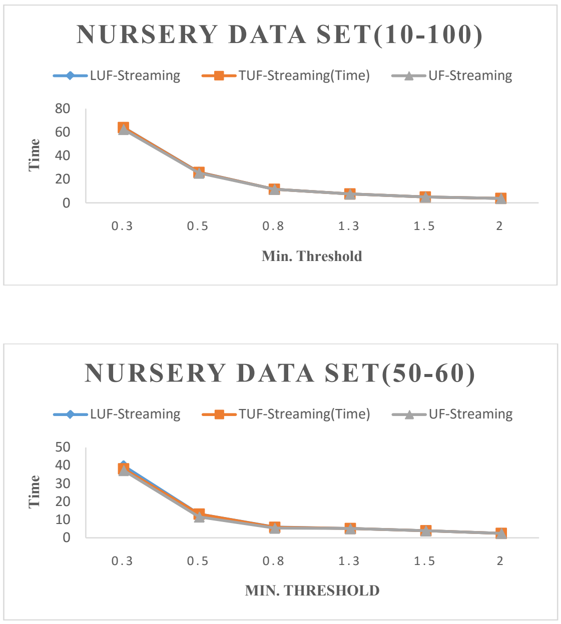


Figure 8: evaluation of different thresholds on Nursery data set (experiment 5)

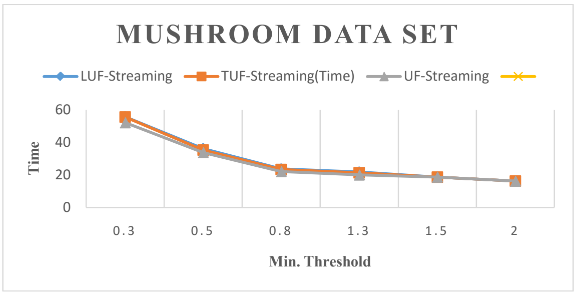


Figure 9: evaluation of different thresholds on Mushroom data set (experiment 5)

5. Related work

The most related previous researches can be categorized as follows.

Approximate dependency search algorithm

We use a search algorithm called AD-Miner to obtain the set of all minimal dependencies. For each attribute such as A from the relational schema R, We search for dependencies in the form to find all possible attributes combinations that are minimal, for the left hand side of dependency. In this search process, we consider a fixed order (such as alphabetical order) for the attributes. The list of attribute that can be used for left hand side of dependency consists of all attributes of R, except A itself. To find the left hand side of dependency, attributes will be selected one by one starting from the bottom of the list. For each selected attribute, such as X, we perform a depth first search to find a set of attributes that contains X, is minimal, and its resulting dependency has an acceptable degree of accuracy.

Key features of AD-Miner algorithm

This algorithm is flexible against exceptions and noisy data and also computes the degree of accuracy of all discovered dependencies. In addition, this algorithm also shows the position of tuples (tuples index in the relationship) that do not hold over each dependency. This feature is especially useful when we want to find exceptions and inconsistent data in a data set. Another important feature of AD-Miner algorithm is that it is Incremental.

6. Conclusion

The results of experiments on data sets show that AD Miner algorithm is one of the most efficient methods among existing incremental and non- incremental algorithms aimed at finding functional dependencies. Another advantage of this method compared with other methods is that it shows tuples that do not follow a single dependency. This feature can be used to detect inconsistent data in a data set. First, we focus our assessment on four introduced algorithms; Evaluations show that among the four proposed algorithms, LUF-streaming algorithm which uses the landmark window model, and TUF-streaming (time) algorithm which uses time-fading window had the best performance. These two algorithms which had the best performance was then compared with existing algorithm, UF-streaming , which uses Sliding window; the result of this comparison show that proposed algorithms are only slightly slower than the UF-streaming algorithm, and this is while UF-streaming uses the sliding window which means that it discards the older data; meanwhile our proposed algorithms use window techniques in which the window size increases with the increase in data, and this means that it provides more information from mining the data stream, which enables users to utilize the older data as well as present data to implement their strategies.

References

[1] Mining association rules in large databases. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 1994), Santiago de Chile, Chile, pages 487{499. Morgan Kaufmann Publishers Inc., 1994.

[2] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. SIGMOD Rec., 29:1{12, May 2000.

[3] Chris Giannella, Jiawei Han, Jian Pei, Xifeng Yan, and Philip S. Yu. Mining frequent patterns in data streams at multiple time granularities. Data Mining: Next Generation Challenges and Future Directions, pages 191{212, 2004.

[4] Carson Kai-Sang Leung and Boyu Hao. Mining of frequent itemsets from streams of uncertain data. In Proceedings of the 2009 IEEE 25th International Conference on Data Engineering (ICDE 2009), Shanghai, China, pages 1663{1670. IEEE Computer Society, 2009.

[5] Carson Kai-Sang Leung, Christopher L. Carmichael, and Boyu Hao. Efficient mining of frequent patterns from uncertain data. In Proceedings of the Seventh IEEE International Conference on Data Mining Workshops (ICDM 2007), Washington, DC, USA, pages 489{494. IEEE Computer Society, 2007.

[6] Chun-Kit Chui, Ben Kao, and Edward Hung. Mining frequent itemsets from uncertain data. In Proceedings of the 11th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2007), Nanjing, China, pages 47{58. Springer-Verlag, 2007.

[7] Carson Kai-Sang Leung, Mark Anthony F. Mateo, and Dale A. Brajczuk. A tree-based approach for frequent pattern mining from uncertain data. In Proceedings of the 12th Paci_c-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2008), Osaka, Japan, pages 653{661. Springer-Verlag, 2008.

[8] Man Lung Yiu, Nikos Mamoulis, Xiangyuan Dai, Yufei Tao, and Michail Vaitis. E_cient evaluation of probabilistic advanced spatial queries on existentially uncertain data. IEEE Transactions on Knowledge and Data Engineering, 21:108{122, January 2009.

[9] Xiangyuan Dai, Man Lung Yiu, Nikos Mamoulis, and Michail Vaitis. Probabilistic spatial queries on existentially uncertain data. In

Proceedings of the 9th International Symposium on Spatial and Temporal Databases (SSTD 2005), Angra dos Reis, Brazil, pages 400{417.Springer-Verlag, 2005.

[10] Chun-Kit Chui and Ben Kao. A decremental approach for mining frequent itemsets from uncertain data. In Proceedings of the 12th Pacific- Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD 2008), Osaka, Japan, pages 64{75. Springer-Verlag, 2008.

[11] George A. Mihaila, Ioana Stanoi, and Christian A. Lang. Anomaly-free incremental output in stream processing. In Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM 2008), Napa Valley, California, USA, pages 359{368. ACM, 2008.

[12] Anamika Gupta, Vasudha Bhatnagar, and Naveen Kumar. Mining closed itemsets in data stream using formal concept analysis. In Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2010), Bilbao, Spain, pages 285{296. Springer- Verlag, 2010.

[13] Nan Jiang and Le Gruenwald. Research issues in data stream association rule mining. SIGMOD Rec., 35:14{19, March 2006.

[14] Carson Kai-Sang Leung and Boyu Hao. Mining of frequent itemsets from streams of uncertain data. In Proceedings of the 2009 IEEE 25th International Conference on Data Engineering (ICDE 2009), Shanghai, China, pages 1663{1670. IEEE Computer Society, 2009.

[15] Mahmood Deypir, Mohammad Hadi Sadreddini, An Efficient Algorithm for Mining Frequent Itemsets Within Large Windows Over Data Streams, International Journal of Data Engineering (IJDE), Volume (2) : Issue (3) : 2011.

[16] Li H, Lee S, Shan M. “An efficient algorithm for mining frequent itemsets over the entire history of data streams”. Proceedings of the first international workshop on konwledge discovery in data streams, Pisa, Italy, 2004.

[17] Carson Kai-Sang Leung and Fan Jiang, Frequent Pattern Mining from Time-Fading Streams of Uncertain Data, LNCS 6862, pp. 252–264, 2011.

[18] Mannila, H. and R¨aih¨a, K.-J. (1991). The design of relational databases. Addison-Wesley.

[19] Brockhausen, P. (1994). Discovery of functional and unary inclusion dependencies in relational databases. Master’s thesis, University Dortmund, Informatik VIII. In German.

[20] Anforderungen An , Lehrstuhl Viii , Lehrstuhl Viii , Siegfried Bell , Siegfried Bell Peter Brockhausen , Peter Brockhausen , Fachbereich Informatik , Fachbereich Informatik, Fachbereich Informatik , Fachbereich Informatik , Fachbereich Informatik, Discovery of Data Dependencies in Relational Databases, Machine Learning: ECML-95: 8th European Conference on Machine Learning, Volume 8, 1995.

[21] P. Bosc, L. Lietard, and O. Pivert, Functional dependencies revisited under graduality and imprecision, pp. 57 - 62, NAFIPS, 1997.

[22] F. Berzal, I.Blanco, D. Sanchez, J.M. Serrano and M.A. Vila, A definition for fuzzy approximate dependencies, Fuzzy Sets and Systems, Vol. 149, No. 2, pp. 105 – 129, 2005.

[23] J. Kivinen and H. Mannila, Approximate inference of functional dependencies from relations, Theoretical Computer Science, Vol. 149, No. 1, pp. 129 – 149, 1995.

[24] J.M. Morrissey, Imprecise information and uncertainty in information systems, ACM Transactions on Information Systems, Vol.8, No. 2, pp. 159–180, 1990.

[25] U. Nambiar, and S. Kambhampati, Answering Imprecise Queries over Autonomous Web Databases, In: Proc. ICDE 2006, 22nd International Conference on Data Engineering, 2006.

[26] S.L. Wang, J.W. Shen, and T.P. Hong, Discovering functional dependencies Incrementally from Fuzzy Relational Databases, in: Proc. English National Conference on Fuzzy Theory and Its Applications, pp. 17 – 24, 2000.

[27] S.L. Wang, J.S. Tsai, B.C. Chang, Mining Approximate Dependencies using partitions on Similarity-Relation based Fuzzy databases, in: Proc. IEEE SMC’99, Vol. 6, PP. 871_875, 1999.

[28] S.L. Wang, J.S. Tsai and T.P. Hong, Discovering functional dependencies from Similarity-based Fuzzy Relational Databases, Intelligent Data Analysis, Vol. 5, No. 2, pp. 131 – 149, 2001.

[29] P. A. Flach and I. Savnik, Database dependency discovery: a machine learning approach, AI communications, Vol. 12, No. 3, pp. 139 – 160, 1999.

[30] H. Mannila and K.J. Raiha. Algorithms for inferring functional dependencies from relations. DKE, Vol. 12, No. 1, pp. 83-99, 1994.

[31] S. Lopes, J.M. Petit, L. Lakhal, Efficient Discovery of Functional Dependencies and Armstrong Relations, in: Proc. ICDT 2000, the 7th International Conference on Extending Database Technology: Advances in Database Technology, Vol. 1777, pp. 350 – 364, 2000.

[32] C. Wyss, C. Giannella, and E. Robertson, FastFDs, A Heuristic-Driven Depth-First Algorithm for Mining Functional Dependencies from Relation Instances, Data Warehousing and Knowledge Discovery, Vol. 2114, No. 1, pp. 101 – 123, 2001.

[33] Y. Huhtala, J. Karkkainen, P. Porkka and H. Toivonen, TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. The Computer Journal. Vol. 42, No. 2, pp. 100 – 111, 1999.

[34] Xuan Hong Dang, Kok-Leong Ong, and Vincent Lee, An Adaptive Algorithm for Finding Frequent Sets in Landmark Windows, Springer-Verlag Berlin Heidelberg, pp. 590–597, 2012.

[35] S.M. Fakhr Ahmad, M. Zolghadri Jahromi, M.H. Sadreddini, A new incremental method for discovery of minimal approximate dependencies using logical operations, Journal of Intelligent Data Analysis, Volume 12 pages.607-619, 2008.