TF-SProD: Time Fading based Sensitive Pattern Hiding in Progressive Data

Shubham Bhatia and **Durga Toshniwal**
Department of Computer Science and Engineering
Indian Institute of Technology, Roorkee
Uttarakhand - India
{shubham1695@gmail.com, durgatoshniwal@gmail.com}

Abstract

In the present era, there is a tremendous generation of transactional data due to e-commerce activities and traditional retail sales. Different organizations want to analyze the data generated by them collaboratively to find some existing trends and patterns in the global transactional data, which may further lead to enhanced strategies for business sales. But this may lead to the disclosure of some patterns which are sensitive to an organization. In order to avoid this, privacy preserving data mining is used. The existing techniques for hiding sensitive patterns in transactional data consider the data as static data. Static data means that the entire data is present beforehand, and no changes are expected in the data, once the processing starts. But most of the data that is received today is dynamic in nature, meaning that the old data may be considered obsolete while new data is being added. In this paper, a technique is proposed for hiding sensitive itemsets from a set of frequent itemsets in progressive data. The proposed algorithms apply sensitive pattern hiding techniques to progressive data by using a sliding window time fading model. A buffer is used to keep track of the top-k frequent itemsets until a given point in time. Both the frequency count of the frequent items and their time of occurence are used to update the buffer. Experiments have been performed on synthetic datasets, and the results are very promising.

1 Introduction

The basic notion of information privacy is to have control over handling and collecting an individual's data. Collection of data from various sources may have many advantages, but it may also lead to information leakage. To deal with information leakage, methods have been proposed, which are known as Privacy Preserving Data Mining (PPDM) techniques. PPDM techniques work by modifying user's original data. But this modification may lead to reduction of data utility. Thus PPDM methods are designed in a way so as to hide user's data, while maximizing the data utility. These methods deal with withdrawal of knowledge from a huge database while preventing the revelation of sensitive information. The main problem with data mining techniques is that people may infer some information which is sensitive to the user, from non-sensitive information. This problem is widely seen when one is trying to find out frequent itemsets from a set of databases. During such an analysis, one may obtain some frequent itemsets, which may have sensitive information about a particular user or organization. For example, if an organization publishes its transactional information without hiding certain patterns, attackers may infer sensitive information from the non-sensitive patterns. To deal with the aforesaid problem, privacy preserving frequent itemset mining is done. The most common way of preserving privacy is to reduce the support count of the sensitive items, to hide them. This process of hiding the sensitive itemsets by reducing their support counts is known as sanitization of the database. But these techniques come with their challenges, and hence they must be designed keeping in mind that a) the impact on the database due to the changes should be as minimal as possible, and b) a good trade-off should be guaranteed between knowledge discovery and privacy.

Sensitive pattern hiding has been categorized into three different approaches: Border-based approaches, exact approaches, and heuristic-based approach. Border-based approaches and exact approaches are not suitable for very large datasets, because they are challenging to implement, and also they are computationally expensive. As the data size grows, the time taken by these approaches also increases. Thus, heuristic based approaches are used mostly. These approaches decide the outcome based on local optima, hence they are fast and simple to implement, as compared to the exact and border-based approaches.

Most of the above approaches deal with privacy preserving frequent itemset mining on static data, meaning that the entire data is present beforehand and no changes are expected in the data once the data is processed. This paper proposes a technique for privacy preserving frequent itemset mining on progressive data. Progressive data is dynamic data, in the sense that it is updated regularly, and some data may fade away or may be considered obsolete while the new data is being generated.

This paper has been organized into six sections. The first section gives a brief introduction about frequent itemset mining and the problems which occur when frequent itemset mining is done collaboratively, for more than one organizations. The introduction also discusses the different approaches of sensitive itemset mining. The second section elaborates the state of art methods and the work done in the area of sensitive itemset hiding. The third section explores the basic terminologies being used. The fourth section explains the proposed approach. The fifth section deals with the experiments performed and the inference from the results of the experiments. Finally, the sixth section concludes the paper.

2 Related Work

Most of the work done in the field of privacy preserving data mining has been done on static data. For example, Oliveira and Zaiane in [Oliveira and Zaiane, 2002] proposed an approach, in which only in a few scans of the database, multiple itemsets are hidden. In the first scan of the database, an index file is created, which is used to find out sensitive transactions corresponding to any sensitive itemset. In other scans, data is sanitized so that minimal changes is done to non-sensitive patterns. Three algorithms were introduced by them: MinFIA- Minimum Frequent Itemset Algorithm, MaxFIA- Maximum Frequent Itemset Algorithm and IGA- Itemset Grouping Algorithm. In the MinFIA algorithm, sensitive transactions are identified, and then, according to their degree of conflict, they are sorted. Further, from each of these transactions, an item is removed, which is known as the victim item. Victim item is chosen as the item which has minimum support. MaxFIA works in the same fashion, but it selects the victim item as the one with maximum support. IGA works by grouping common items of sensitive itemsets, and the item with minimum support is chosen as victim.

According to the approach proposed by Cheng et al. in [Cheng *et al.*, 2016], the very first step of the process is to store the count of all non-sensitive patterns that a transaction supports. Next, the count of each transaction is stored corresponding to each sensitive item, and these transactions are sorted in accordance to their supports calculated in the first step. Then victim items are removed from transactions. Victim item is the item in sensitive pattern which has maximum support.

A confidence-based approach was proposed by Verykios et al. in [Verykios *et al.*, 2004]. In this approach the hiding of sensitive patterns is done by decreasing the confidence of an association rule because lesser side effects are caused to the sanitized dataset. One limitation of this approach is that it does not guarantee hiding of all sensitive itemsets.

The authors in [Jangra and Toshniwal, 2020] suggest a victim item deletion based sensitive pattern hiding technique, in which each particle of the population consists of n number of sub- particles derived from pre-calculated victim items.

An approach for hiding sensitive patterns without any side effects is suggested in [Surendra and Mohan, 2019]. The authors propose Recursive Pattern Sanitization (RPS) algorithm which hides multiple sensitive itemsets irrespective of their support count and size. This is done in single parse of the closed patterns. The patterns retain the closeness property in the sanitized model, and the model has inherent support for finding frequent itemsets.

Few researches have been done in progressive data. For frequent itemset mining in progressive data, various approaches have been proposed in [Mhatre and Toshniwal, 2010], [Mhatre *et al.*, 2009] and [Thool and Voditel, 2013]. The broad classification of these approaches is counter based algorithms and sketch based algorithms. Counter based approaches generally maintain a counter variable, and increment the counter if the item is seen again. Space saving and lossy counting algorithm are types of counter based approaches. In sketch based approaches, both arrival and departures of an item are kept in mind. Fernandez-Basso et. al. in [Fernandez-Basso *et al.*, 2019] propose a method for finding tendencies in streaming data using Big Data frequent itemset mining. They propose a frequent itemset mining method using sliding windows capable of extracting frequently occurring items from continuous data flows. Yamamoto et. al. in [Yamamoto *et al.*, 2019] propose a hybrid approximation approach for scalable frequent itemset mining in streaming data. They argue that any streaming data algorithm must possess two important properties; (i) the real time property, which means the property to process a large volume of data which arrives continuously at high speed and detect the frequent itemsets simultaneously, and (ii) memory efficiency, which means the property to maintain the frequent itemsets for the entire database, using limited amount of memory. Yun and Lee in [Yun and Lee, 2016] discuss an approach for Incremental mining of weighted maximal frequent itemsets from dynamic databases. They discuss the issues with stream data mining, i.e., the results can be very large scale. Storing these results may be difficult to analyze and all of them may not convey some meaningful information. Thus, they propose algorithm to extract smaller number of itemsets. Most of the existing research works on progressive data, particularly the space saving algorithm has its own disadvantages. The space saving algorithm takes into account only the support count of the frequent itemsets. As and when a new item is received, if the buffer can accommodate it, it is inserted into the buffer, otherwise the element with the minimum support is deleted from the buffer and the new element is inserted. The space saving algorithm deletes the element only on the basis of its support count, but it does not take into account the time at which the element occurred. In this paper, timestamps are also taken into account while deleting an item from the buffer.

Most of the existing research works for hiding sensitive patterns have been developed for static data but not for progressive data. Static data remain constant with time, in the sense that entire data is available at once, and no new items are added / deleted. On the contrary, progressive data is more complex to deal with. Progressive data is a stream of data, with continuous inflow of items. The new incoming data can increase the support count of some items which were not frequent previously, and also remove some items which were frequent at some point of time previously but are not seen in the recent transactions. Thus, there is a continuous upgradation of data. This makes working with such data difficult because tracks have to be maintained about the addition and deletion of items, and how this addition and deletion affects the itemsets already present.

3 BASIC TERMINOLOGIES

3.1 Frequent Itemset Mining

Let X be a set of items, $X = \{x_1, x_2,.....,x_n\}$. Let T be set of transactions that make the database. Each transaction in T is an itemset, i.e., $\forall\ t \in T,\ t \subseteq X$. With each transaction, a unique identifier is associated, which is known as TID. Table 1 shows a sample database with 5 transactions.

TID	Itemsets
T1	ABE
T2	BD
T3	BC
T4	ABD
T5	ABCE

Table 1: Sample Database T

Apriori algorithm[Agrawal *et al.*, 1994] is used for frequent itemset mining from a set of transactions. The apriori algorithm proceeds by identifying individual items which are frequent in the database, and then expanding the frequent items to larger and larger sets, until the itemsets so formed do not satisfy a minimum threshold value. This minimum threshold value is known as minimum support.

Support indicates how often an itemset is seen in the database. The support S of an itemset I with respect to T can be defined as the number of transactions t in T in which I occurs.

$$supp(I) = \frac{|t \in T; t \subseteq I|}{|T|} \tag{1}$$

The confidence of an association rule, $A \implies B$ is defined as the number of transactions that contain A *that also contain B*.

$$conf(A \implies B) = \frac{supp(A \cup B)}{supp(A)} \tag{2}$$

Association rules generated after applying the apriori algorithm satisfy (i) a minimum threshold count, known as support, which is specified by the user, and (ii) a minimum confidence value.

When apriori is applied to the database in Table 1, with a minimum support count of 2(support of 40%) the frequent itemsets are generated, as described in Table 2.

Itemset	Support Count
AB	3
AE	2
BC	2
BD	2
BE	2
ABE	2

Table 2: Frequent Itemsets

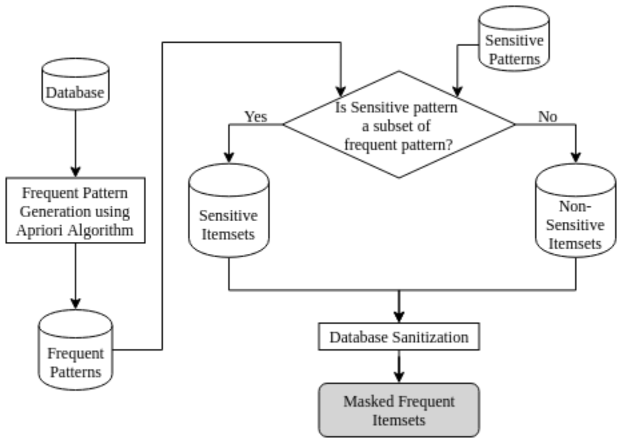

Figure 1: Block diagram for proposed work

3.2 Hiding Sensitive Itemsets

In collaborative data mining, when two or more companies publish their data together, which can be further analysed, there is a chance that some private information of a company can be leaked by mining the frequent itemsets. Thus every company hides some information which is sensitive to it, by using privacy preserving data mining.

Let X be a set of items, $X = \{x_1, x_2,.....,x_n\}$ which has to be hidden from the database. With a given minimum support s, the problem reduces to the transformation of the set of transactions T in the database to T' such that the support of any itemset $x_i < s$, and $|\ supp_T\ (x_i)\ \text{-}\ supp_{T'}\ (x_i)\ |$ is minimum.

3.3 Period of Interest

For progressive data, a fixed sized sliding window is used. This is also denoted as Period Of Interest(POI). The transactions which have timestamp falling into POI contribute to frequent itemset mining. For a particular POI, frequent itemsets are found and are mapped to a buffer with their frequency count and timestamp. Also, obsolete items are removed in parallel, as and when the buffer reaches its maximum capacity.

4 PROPOSED WORK

The flow diagram of the proposed work is represented by Figure 1. In the proposed approach, firstly the frequent itemsets are mined from the continuous stream of data by applying the apriori algorithm. Secondly, the sensitive itemsets that are supposed to be hidden, are masked by reducing the support count of victim items below the minimum threshold. The database obtained after masking the sensitive itemsets is known as sanitized database. Finally, the itemsets obtained thereafter are mapped onto a buffer with their frequency count and their timestamp of occurrence as shown in Figure 2. Further, the subsections discuss the steps of the proposed approach in detail.

4.1 Frequent Pattern Generation

Apriori algorithm is applied for generating frequent patterns from the continuous stream of transactions. The apriori algorithm takes a minimum threshold value as input. This value of the minimum threshold specifies the minimum number of times an item has to be present in the transactions, so that is is marked as a frequent item. The Apriori algorithm firstly finds individual frequent items in the entire set of transactions, and then it proceeds by forming sets of items called frequent patterns, until no further itemset can be found that satisfies the minimum support criteria.

4.2 Differentiating sensitive and non-sensitive itemsets

The frequent patterns found by Apriori algorithm are further divided into two sets: sensitive itemsets and non-sensitive itemsets. This division is done on the basis of pre-defined sensitive patterns which is provided by the organisation whose data has to be masked. If any sensitive pattern is found to be a subset of a frequent pattern, then that particular frequent pattern is added to the set of sensitive itemsets, otherwise the frequent pattern is added to the set of non-sensitive itemsets.

4.3 Database Sanitization

Firstly, a victim item is to be chosen for each sensitive itemset. This paper proposes two methods for selecting victim items for sensitive pattern hiding in progressive data:

- TF-SProDmax(**T**ime **F**ading based **S**ensitive pattern hiding in **Pro**gressive **D**ata *max*): In this method, the item with maximum support is chosen as the victim item.
- TF-SProDmin(**T**ime **F**ading based **S**ensitive pattern hiding in **Pro**gressive **D**ata *min*): In this method, the item with minimum support is chosen as the victim item.

After selecting the victim items, the support count of the sensitive itemsets is reduced below the minimum threshold. This is done so that when frequent pattern mining is applied on sanitized data, the sensitive patterns are never discovered by anyone.

4.4 Adding sanitized items to buffer

Once the database is sanitized, a set of final frequent itemsets is obtained. These frequent itemsets are to be added to the buffer. The buffer contains a list of overall frequent items with their timestamp and frequency. The count is used to discard less frequent(obsolete) items, once the buffer is full. It may happen that an item has less frequency in the buffer because it was added recently to the buffer. In such a case, timestamp is used so that an item can be deleted from the buffer considering not only its frequency, but also the time at which it was added.

Figure 2 shows how the items are added to the buffer.

- If the item is already present in the buffer, its frequency is incremented by 1, and its timestamp is updated to the current timestamp.
- If the item is not present in the buffer, but the buffer has space, then the item is simply added to the buffer with an initial frequency of 1.
- If the item is not present in the buffer, and the buffer is full, the item with least frequency and oldest timestamp is removed from the buffer.

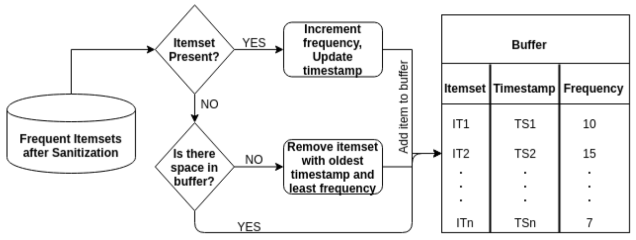
Frequent Itemsets after Sanitization
Itemset Present?
YES
NO
Increment frequency, Update timestamp
Is there space in buffer?
NO
Remove itemset with oldest timestamp and least frequency
YES
Add item to buffer
Buffer
Itemset
Timestamp
Frequency
IT1
TS1
10
IT2
TS2
15
ITn
TSn
7

Figure 2: Adding frequent items to buffer

5 EXPERIMENTS AND DISCUSSIONS

The experiments were conducted to test the misses cost of the sanitized dataset, and comparison has been made on the two proposed algorithms: TF-SProD*max* and TF-SProD*min*. Also, to show the importance of timestamps, accuracy of TF-SProD, which is a timestamp based method is compared with respect to algorithm without timestamp. All the experiments were performed using Python 3.7 on 64-bit Ubuntu machine with 80 GiB RAM.

5.1 Simulation of progressive data

To simulate the effect of progressive data, the entire dataset has been divided into chunks. Each chunk denotes a new incoming set of transactions at a particular point in time. Sensitive patern hiding is applied to i^{th} chunk, then the sanitized itemsets are stored in a buffer. The process is carried out similarly for $i+1^{th}$ to n^{th} chunks.

5.2 Dataset Used

The datasets used for the experiments was generated by IBM data generator. Different datasets are generated, and results are varied according to the input parameters.

The datasets used have specifications as shown in Table 3

Dataset	Number of Transactions	Chunk Size
Dataset 1	176324	400
Dataset 2	352648	400
Dataset 3	705296	400
Dataset 4	1057944	400

Table 3: Dataset Configuration

5.3 Generation of sensitive itemsets

Since sensitive itemsets of an organization depend upon the information which is private to the organisation, they are not externally generated by any algorithm, rather the sensitive information has to be provided by the organisation itself. Thus, while hiding the sensitive itemsets, they have to be explicitly specified. Since the data being used in this paper is synthetic, the sensitive itemsets are also synthetically generated. Some of the frequent itemsets are randomly chosen from the entire dataset and they are marked as sensitive itemsets.

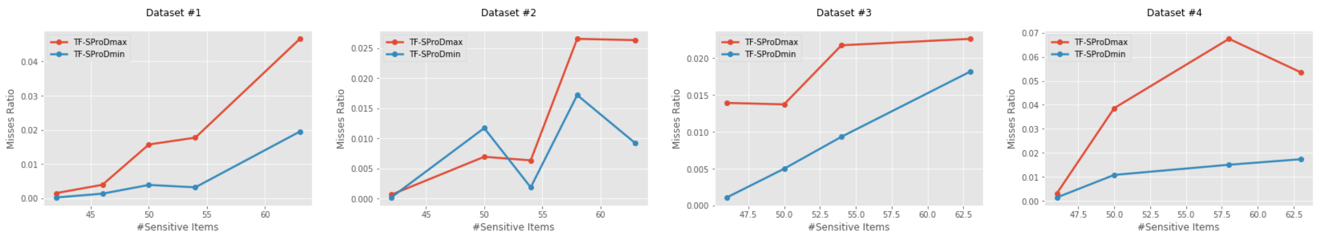

Figure 3: Variation of misses ratio with number of sensitive items, keeping minimum support constant for different datasets

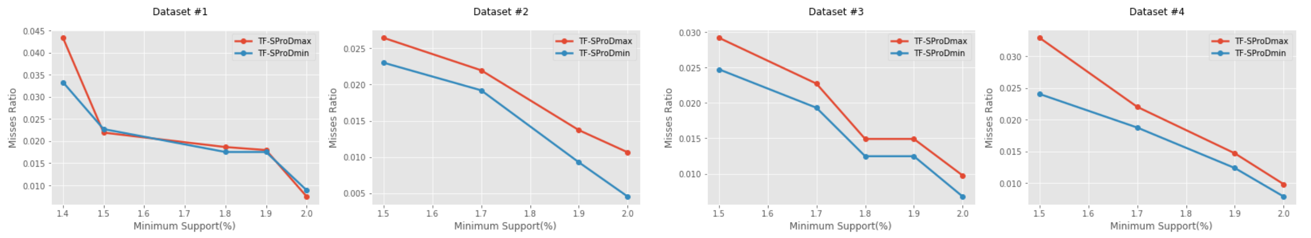

Figure 4: Variation of misses ratio with minimum support, keeping number of sensitive items constant for different datasets

5.4 Performance metrics

Let S be a set of sensitive itemsets, $NSI_{original}$ be the non- sensitive frequent itemsets in the original database and $NSI_{sanitized}$ be the non- sensitive frequent itemsets in sanitized database. Misses cost is defined as the number of non-sensitive itemsets, that were mistakenly hidden by the algorithm. Misses cost is defined by equation 3.

$$MC = |NSI_{\text{original}} - NSI_{\text{sanitized}}| \tag{3}$$

Lesser the misses cost, the more efficient the algorithm is.

Misses ratio is the ratio of misses cost to $NSI_{original}$. Misses ratio is defined by equation 4.

$$MR = \frac{MC}{NSI_{\text{original}}} \tag{4}$$

5.5 Experimental Results

Comparison of TF-SProdmax and TF-SProdmin

Experiments have been performed to analyze the effect of:

- Choosing the element with the maximum support as the victim item (TF-SProD*max*)
- Choosing the element with the minimum support as the victim item (TF-SProD*min*)

The experiment results are based on two types of variations:

1. Finding misses ratio by keeping the support count constant while varying the number of sensitive items. [Fig. 3]
2. Finding misses ratio by keeping the number of sensitive items constant while varying the support count. [Fig. 4]

The Variation of misses ratio by changing the number of sensitive items, keeping minimum support constant for different datasets is illustrated in Figure 3. The minimum support was kept at 2%. As can be inferred from the graph, a general trend can be seen that by increasing the number of sensitive items, there is an increase in the misses ratio. This can be explained as, the increase in the number of sensitive items will be incurred at the cost of more and more items(victim items) being removed from the transaction set, to reduce the count of sensitive items below minimum support. This may, in turn, mark some items as sensitive, which were not sensitive before sanitization. Hence there is an increase in the misses ratio.

Also, it can be seen that TF-SProD*min* performs better than TF-SProD*max*, in the sense that it has lower misses ratio that TF-SProD*max*, at all points. This can be explained as: since in TF-SProD*min* the item with minimum support is chosen as the victim item, hence less number of transactions are to be altered during the sanitization of the database. Thus, the misses ratio is less for TF-SProD*min*.

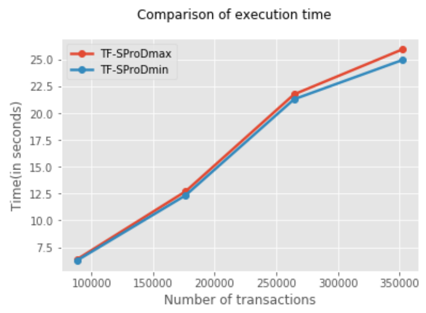

Figure 5: Comparison of execution time for TF-SProD*max* and TF-SProD*min*

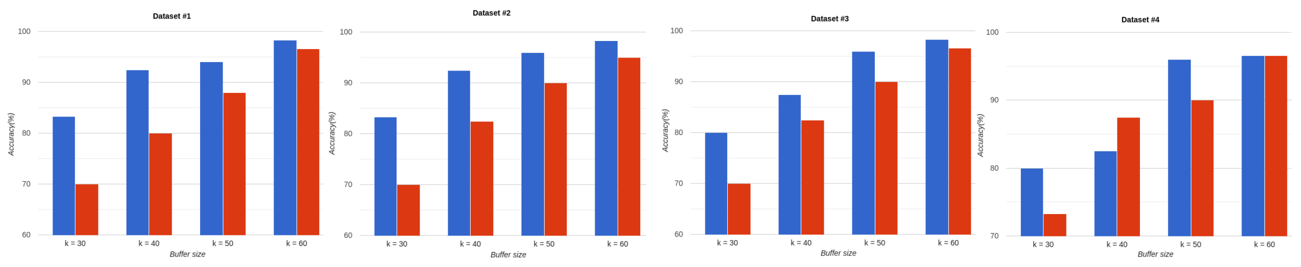

Figure 6: Bar plots showing differences between accuracy of sensitive pattern hiding by using timestamp (TF-SProD) shown by blue bars and without using timestamps shown by red bars for all the four datasets.

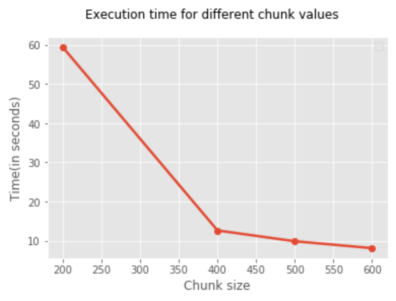

Figure 7: Execution time for various chunk sizes

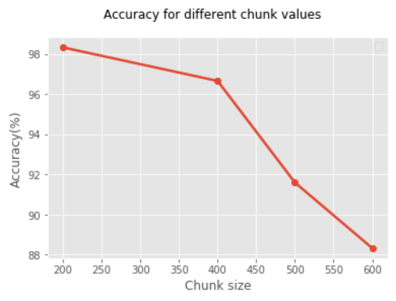

Figure 8: Accuracy for various chunk sizes

Figure 4 shows the Variation of misses ratio by changing the minimum support, keeping the number of sensitive items constant for different datasets. The number of sensitive items is kept as 84. It can be seen that with an increase in support count, the misses ratio decreases. This is because as the support count is increased, fewer items qualify to be frequent items, and hence fewer transactions are altered during database sanitization. Hence the misses ratio can be seen to decrease gradually.

Again, it can be seen that TF-SProD*min* performs better than TF-SProD*max*, in the sense that it has lower misses ratio that TF-SProD*max*, at all points, for the same reason as stated in above paragraph.

Figure 5 compares the execution time(in seconds) for TF-SProD*min* and TF-SProD*max*. The support was kept at 2%, the number of sensitive items was kept as 84, and the chunk size was 400. It can be seen that TF-SProD*min* has slightly less execution time for the same number of transactions. This is because, in TF-SProD*min*, the item with minimum support is chosen as victim item. Thus less number of transactions are to be altered to hide the sensitive patterns, hence the execution time is less.

Accuracy of TF-SProD

The accuracy of the proposed approach can be measured by counting the number of itemsets that were stored in the buffer at the end of the algorithm, and finding the number of items that were actually frequent when the entire dataset is considered as one single stream of data. The accuracy is defined as:

$$Accuracy = \frac{\text{Number of items that were actually frequent}}{\text{Total number of items in the buffer}}$$

The accuracy is found out by comparing the items that were stored in the buffer after the last chunk was processed, with their support count if the entire dataset was processed as a single stream.

The accuracy of pattern hiding algorithm by using timestamps in the buffer(TF-SProD), as compared to pattern hiding without using timestamp is shown in Figure 6. The number of sensitive patterns were taken as 10. The minimum support count was kept at 2%. The graph analyses the accuracy of the two algorithms by varying the size (k) of the buffer for different datasets. It can be seen that the accuracy of TF-SProD outperforms the accuracy of pattern hiding without timestamps. Particularly for small buffer sizes, the difference in the accuracy is very prominent. This is because for small buffer sizes, items in the buffer have to be replaced quite frequently, as and when new frequent itemsets arrive. When we use timestamp as a measure to remove the item, only obsolete items are deleted. But when we do not use timestamps and use only support count to remove items from the buffer, new items may be deleted from the dictionary, because they were added recently and had very less support count. Thus TF-SProD proves to be better in case of accuracy of the items.

Determining chunk size

The chunk value chosen for the experiments was 400. Figure 7 shows the execution time of TF-SProD*min* for different chunk sizes and Figure 8 shows the accuracy of TF-SProD*min* for various chunk sizes. From the figures, it is clear that the run time for chunk size 200 is maximum, but accuracy for the same chunk is the best. Similarly for other chunk sizes, as we increase the chunk size, the runtime reduces, but at the cost of reduction of accuracy. Thus, a trade-off between accuracy and chunk size is required, so that the algorithm can be run with less execution time and good accuracy. For that purpose, the chunk size in the algorithms has been chosen as 400.

6 CONCLUSION

Most of the work in privacy preserving frequent pattern mining has been done on static data. However, most of the real world data is progressive in nature. Thus, in this research work, two approaches for sensitive pattern hiding in progressive data: TF-SProD*max* and TF-SProD*min* are proposed. Extensive experiments have been performed on synthetic datasets. The performance of TF-SProD*min* is seen to be better, both in terms of execution time and misses ratio.

To simulate a continuous flow of data, sliding window time fading model has been used, which keeps into account the frequency count of the frequent patterns, as well as their timestamp. The importance of timestamp can be realised as the TF-SProD outperforms the algorithm without timestamp in terms of accuracy. The timestamp ensures that a recently added item with less frequency is not deleted, rather deletion of an item from the buffer is based on both frequency and its time of addition into the buffer.

The research area of privacy preservation in progressive data has great potential being an untouched area. A lot of research can happen in this domain such as parallelization of the algorithms using Hadoop MapReduce Framework, etc.

References

[Aggarwal and Philip, 2008] Charu C Aggarwal and S Yu Philip. *Privacy-preserving data mining: models and algorithms*. Springer Science & Business Media, 2008.

[Agrawal and Srikant, 2000] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 439–450, 2000.

[Agrawal *et al.*, 1994] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[Amiri, 2007] Ali Amiri. Dare to share: Protecting sensitive knowledge with data sanitization. *Decision Support Systems*, 43(1):181–191, 2007.

[Cheng *et al.*, 2016] Peng Cheng, John F Roddick, Shu-Chuan Chu, and Chun-Wei Lin. Privacy preservation through a greedy, distortion-based rule-hiding method. *Applied Intelligence*, 44(2):295–306, 2016.

[Fernandez-Basso *et al.*, 2019] Carlos Fernandez-Basso, Abel J Francisco-Agra, Maria J Martin-Bautista, and M Dolores Ruiz. Finding tendencies in streaming data using big data frequent itemset mining. *Knowledge-Based Systems*, 163:666–674, 2019.

[Jangra and Toshniwal, 2020] Shalini Jangra and Durga Toshniwal. Vidpso: Victim item deletion based pso inspired sensitive pattern hiding algorithm for dense datasets. *Information Processing & Management*, 57(5):102255, 2020.

[Lee *et al.*, 2004] Guanling Lee, Chien-Yu Chang, and Arbee LP Chen. Hiding sensitive patterns in association rules mining. In *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, pages 424–429. IEEE, 2004.

[Lin *et al.*, 2015] Chun-Wei Lin, Tzung-Pei Hong, Kuo-Tung Yang, and Shyue-Liang Wang. The ga-based algorithms for optimizing hiding sensitive itemsets through transaction deletion. *Applied Intelligence*, 42(2):210–230, 2015.

[Mhatre and Toshniwal, 2010] Amruta Mhatre and Durga Toshniwal. Hiding co-occurring sensitive patterns in progressive databases. In *Proceedings of the 2010 EDBT/ICDT Workshops*, pages 1–5, 2010.

[Mhatre *et al.*, 2009] Amruta Mhatre, Mridula Verma, and Durga Toshniwal. Extracting sequential patterns from progressive databases: A weighted approach. In *2009 International Conference on Signal Processing Systems*, pages 788–792. IEEE, 2009.

[Oliveira and Zaiane, 2002] Stanley RM Oliveira and Osmar R Zaiane. Privacy preserving frequent itemset mining. In *Proceedings of the IEEE international conference on Privacy, security and data mining-Volume 14*, pages 43–54. Australian Computer Society, Inc., 2002.

[Oliveira and Zaïane, 2003] Stanley RM Oliveira and Osmar R Zaïane. Protecting sensitive knowledge by data sanitization. In *Third IEEE International Conference on Data Mining*, pages 613–616. IEEE, 2003.

[Radhakrishna *et al.*, 2015] Vangipuram Radhakrishna, PV Kumar, and V Janaki. A survey on temporal databases and data mining. In *Proceedings of the The International Conference on Engineering & MIS 2015*, pages 1–6, 2015.

[Surendra and Mohan, 2019] H Surendra and HS Mohan. Hiding sensitive itemsets without side effects. *Applied Intelligence*, 49(4):1213–1227, 2019.

[Thool and Voditel, 2013] Manisha Thool and Preeti Voditel. Association rule generation in streams. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(5), 2013.

[Verykios *et al.*, 2004] Vassilios S Verykios, Ahmed K Elmagarmid, Elisa Bertino, Yücel Saygin, and Elena Dasseni. Association rule hiding. *IEEE Transactions on knowledge and data engineering*, 16(4):434–447, 2004.

[Yamamoto *et al.*, 2019] Yoshitaka Yamamoto, Yasuo Tabei, and Koji Iwanuma. Parasol: a hybrid approximation approach for scalable frequent itemset mining in streaming data. *Journal of Intelligent Information Systems*, pages 1–29, 2019.

[Yun and Lee, 2016] Unil Yun and Gangin Lee. Incremental mining of weighted maximal frequent itemsets from dynamic databases. *Expert Systems with Applications*, 54:304–327, 2016.