University of Manitoba

Vertical Mining Of Time Fading Data Streams

Tayler Frederick (7640955)

Evan Spearman (7690987)

Morgan Epp (7691023)

Dr. Carson Leung

Comp 4710 - Data Mining

December ~~10~~, 2015

Table of Contents

1 Introduction

In recent years, there has been much interest in what are referred to as vertical mining algorithms, for finding frequent itemsets in market basket databases[1]. While these algorithms have been shown to outperform horizontal frequent pattern mining algorithms in certain circumstances, much of the research done applies only to static, or unchanging datasets. We aim to introduce vertical mining algorithms to the domain of data stream mining using a time fading to approach to determine if those results hold true there as well.

2 Vertical Mining Background

2.1 VIPER

The VIPER algorithm is an important algorithm in this category as it doesn't put any special requirements on the data it is processing[1], and it is more efficient than many of the other vertical algorithms proposed before it[1]. Vertical frequent pattern mining algorithms differ from more traditional horizontal frequent pattern mining algorithms in that they are designed to work on data where the particular transactions that each itemset is in are kept track of instead of instead of keeping track of which itemsets are in each transaction. In the case of VIPER, each itemset is represented by a bit vector the length of the number of transactions in the database, where each bit is 1 if the corresponding transaction contains that itemset, and 0 if it doesn’t[1]. These bit vectors can easily generate bit vectors representing the union of two itemsets by applying a bitwise and operation between the bit vectors of the sub-itemsets in what is referred to as the FANGS algorithm[1]. VIPER also includes a number of other optimizations with respect to candidate generation to make the algorithm more efficient[1].

2.2 Eclat

Eclat is another implementation of vertical frequent pattern mining, and it ’s biggest difference from VIPER is that each itemset is represented by a transaction id set containing only the transactions that each specific item is in. This means that unlike VIPER, Eclat doesn’t explicitly keep track of transactions that an itemset is not in, but rather implicitly. It should be seen here that due to only storing the transactions that an item is in, we save memory over VIPER when the data is sparse. We can find the transactions that multiple items are in by taking the intersection of the subsets that make up those transactions[4]. For instance if we wanted to find all the transactions that contain A and B, then we’d take the intersection of A with B to produce the transaction id set AB. Acquiring the frequency is also quite trivial by taking the size of the transaction id set[4].

3 Time Fading Background

In standard data mining algorithms, the number of items and transactions in a database are limited to a fixed and comparatively small size. However, in real life this will not always be the case as new data may be added, possibly at high velocity, over a period of time. There is a clear need for being able to differentiate the importance of this more recent data over the historical data and the Time Fading approach is one method to do this[5]. Time Fading allows us to differentiate the importance of data for transactions that happened earlier on versus more recent data[5]. As demonstrated in [5], this can be achieved by separating transactions into batches and applying a fade factor (a value between 0 and 1 inclusive) to the current frequency counts. When more batches come into the database the fade factor is once again applied to the older data, thus giving frequency results with more weight to recent data. Time Fading has mostly been applied to horizontal, tree-based algorithms before[5], and as such provide motivation to combine them with vertical data mining.

4 Time Fading VIPER

We make one assumption about the nature of the streams of data being received, which is that streams are not necessarily able to be processed in real time as the arrive, which could easily be the case in an environment where data arrives frequently, but the computing power required to process the data in real time is not available. Instead our time fading VIPER modification is designed to work in a scenario where a number of batches are already known, but there is still a desire to analyze how the frequency of itemsets change as older data in older batches of transactions becomes less and less relevant.

The first algorithm we propose is an adaption on the VIPER algorithm specified in [1]. The first challenge in adapting VIPER to work with the time-fading model is finding a method of representing the time fading factor as the algorithm progresses through each batch of transactions. The support of an item in the time fading model, where s[] is the support for the item in a particular batch, n is the total number of batches, and f is the fade factor is $\sum_{(i=0 \text{ to } n-1)}(s[i] * f^{n-(i+1)})$ [2]. The VIPER algorithm calculates the support for an item set by summing a bit vector for each itemset[1]. Because of this, we need to know the per-batch fade factor ($f^{n-(i+1)}$ from the above summation) quite frequently as if we wish to fade the relevance of a particular batch of transactions over time, this value must be multiplied by the bit value for each transaction in a particular batch when summing the bit vector. Because of the obvious space reduction (for dense data sets) gained from storing data in bit vectors representing whether a certain itemset is present in each transaction, we want to retain the bit vector data structure. This means however, that we cannot cache this value with the bit vectors themselves, as more than one bit is required to store the per-batch fade factor.

What we propose instead to solve this problem is to keep track of a fade vector. This fade vector will only need to be the length of the number of transactions and will keep track of the per-batch fade factor that is applied to each transaction. For example, the following set of batches of transactions with a fade factor of 0.5:

Batch	Transaction #	Items
1	1	1, 2, 3, 4, 5
1	2	2, 3, 5
2	3	1, 2, 4, 5
2	4	1, 2, 3, 4, 5
3	5	1, 2, 4, 5
3	6	2, 3, 4

Would be represented by the following bit-vectors:

1: 101110
2: 111111
3: 110101
4: 101111
5: 111110

And the fade vector:

(0.25, 0.25, 0.5, 0.5, 1, 1)

By keeping data in these data structures, VIPER's FANGS algorithm for combining two itemsets by taking bitwise and value of the bit vectors representing each item set[1] can proceed as it normally would. The main difference is that in actually summing the bit vectors to get the final support value for each itemset, the corresponding values in the bit vector and fade vector are multiplied together, and the products added up to produce the final support value.

With these data structures established, the algorithm is very similar to the original specification of VIPER. It should be noted that our implementation of time-fading VIPER omitted the creation of the DAG structure, as it is not an out of core implementation and doesn't need to keep track of which bit vectors must be read from disk, as they're all stored in main memory to begin with. The other changes we have made all have to do with the actual calculations of support values, as we will explain in the analysis section, these changes should not affect the time complexity of the algorithm.

VIPER behaves differently when it's finding frequent 1-itemsets, frequent 2-itemsets, and k-frequent itemsets where $k > 2$ [1]. Each of these phases in the algorithm must be adapted to work with the time fading model slightly differently. In the first round, this is fairly trivial. Time Fading VIPER goes through each 1-itemset and sums its bit vector against the fade vector as described above to get the support for that itemset with respect to the batches that are known. As would be expected, non-frequent itemsets are pruned, and disregarded in subsequent rounds.

Finding frequent 2-itemsets in VIPER is quite different from the other iterations. Instead of using the FANGS algorithm to create bit vectors for 2-itemsets from the 1-itemsets, the frequent 1-itemsets are transformed into a horizontal representation of each transaction, these transactions are processed by generating all pairs in a transaction, and adding keeping track of the total support for each pair over the frequent 1-itemsets by accumulating the number of times a pair appears in a 2d triangular array indexed by the members of each pair[1]. Time Fading VIPER modifies this step slightly, in that instead of accumulating the total number of appearances of a pair, the 2d array accumulates value the values from the fade vector for the transaction where the pair is generated. Like with normal VIPER, frequent 2-itemsets are then generated by iterating over this array, and recording the pairs with support >= minsup, where the support value for the pair is the value in the array at arr[frequent 1-item a, frequent 1-item b], where frequent 1-item a < frequent 1-item b[1]. Time Fading VIPER then proceeds with generating the bit vectors of the frequent 2-itemsets just like in normal VIPER[1].

Time Fading VIPER's k > 2 iterations are very similar to that of normal VIPER. It still uses the same FORC algorithm to generate candidates[1]. The difference comes in the actual determination of a candidate itemset being frequent. To find the support of an itemset, the bit-vector of the itemset is summed against the fade vector as was described before.

Applying this algorithm to the transaction database described above with fade factor = 0.5, and minsup = 2, the following frequent itemsets and their support values are produced:

{1}:2.25, {2}:3.5, {3}:2, {4}:2.75, {5}:2.5, {1, 2}: 2.25, {1, 5}:2.25, {2, 3}:2, {2, 4}:2.75, {2, 5}:2.5, {1, 2, 5}:2.25

5 Time Fading Eclat

As previously mentioned in section 2.2, Eclat is quite similar to VIPER (vertical mining), but instead of storing bit vectors to keep track of which transactions an itemset appears in, the transactions are stored as elements in sets, and the set intersection operation is used in place of the AND operation to combine two itemsets. Time Fading Eclat, uses the same fade vector structure that Time Fading VIPER uses to keep track of the fade factor of different transactions. There are a few differences due to the storing of transaction id's such as Eclat being able to use the transactions as an index into the fade vector. This allows it to calculate the support more efficiently than VIPER which must check each and every bit.

An example of Time Fading Eclat, using the same transaction database as the Time Fading VIPER example above:

Batch	Transaction #	Items
1	1	1, 2, 3, 4, 5
1	2	2, 3, 5
2	3	1, 2, 4, 5
2	4	1, 2, 3, 4, 5
3	5	1, 2, 4, 5
3	6	2, 3, 4

Would be represented by the following TId sets:

1: t1, t3, t4, t5
2: t1, t2, t3, t4, t5
3: t1, t2, t4, t6
4: t1, t3, t4, t5, t6
5: t1, t2, t3, t4, t5

And the fade vector:

(0.25, 0.25, 0.5, 0.5, 1, 1)

6 Analysis

6.1 Theoretical Analysis of Time Fading VIPER

Ultimately, Time Fading VIPER should have the same time complexity of the standard VIPER algorithm. This is because all of the changes that the Time Fading Viper specifies are related to the summation of the bit vectors, which is done in O(n) time with respect to the number of transactions. The only difference in this case is that the values of the fade vector are multiplied by the bit values, and since these can be looked up in constant time assuming the fade vector is stored in an array, this will not have an impact on time complexity.

6.2 Experimental Results

Sample output of our tests of the vertical Time Fading Algorithms vs Time Fading Apriori can be found in 1k5Loutput.txt and 10k5Loutput.txt. Figures below are from those data and can be found in results.xlsx.

Figure 1. Time Fading Algorithm Run Times On 1k5L.txt

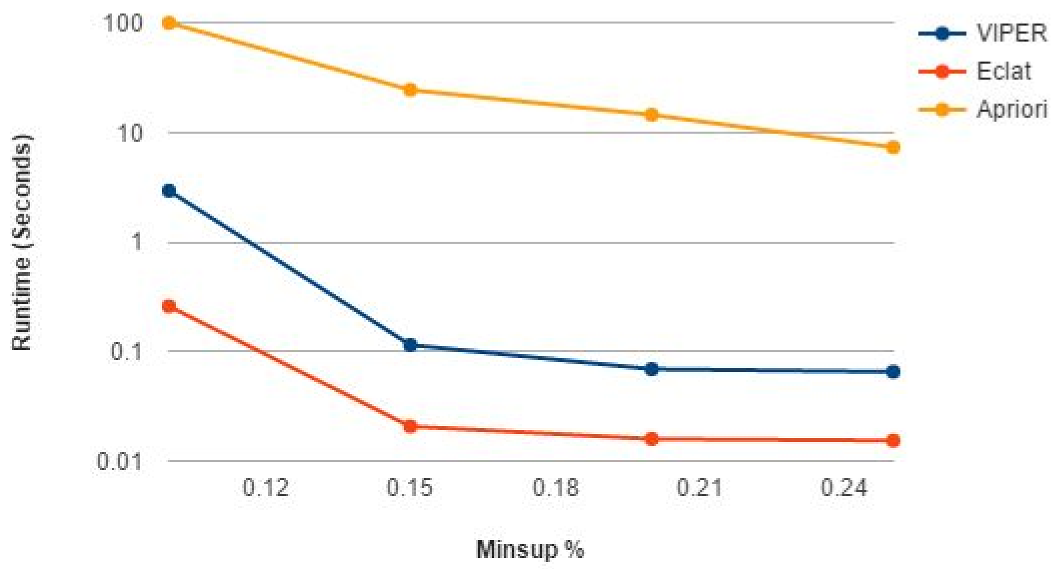
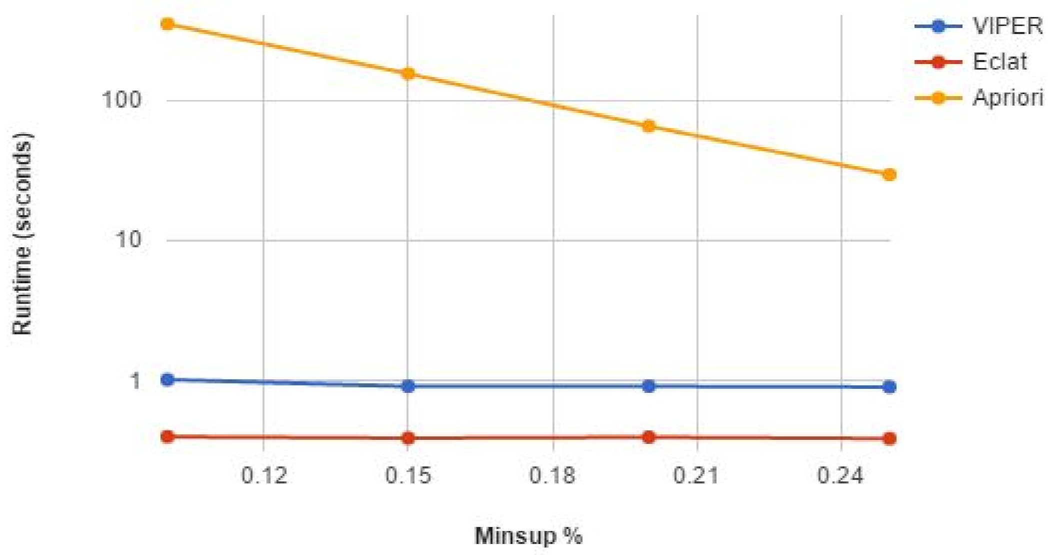


Figure 2. Time Fading Algorithm Run Times On 10k5L.txt



For our experimental results, we ran Time Fading VIPER, Time Fading Eclat, and a Time Fading Apriori implementation against the 1k5L.txt and 10k5L.txt datasets with various minsup percentages (0.1%, 0.15%, 0.2%, 0.25%). We divided the data into batches of 128 transactions for 1k5L.txt, and batches of 1000 transactions for 10k5L.txt. The fade factor used was 0.5. We chose the minsup percentages that we did to analyze the data in such a way that meaningful results were produced. Going too much higher or lower with the minsup percentage caused either nearly all the itemsets in the dataset to be considered frequent, or none of them. This can be adjusted to better match different densities of data.

The results themselves can be found in tabular form in results.xlsx. The runtime results we found seemed to match with our expectations of the algorithms. Time Fading VIPER and Time Fading Eclat were able to reach a consistently low run-time, even on larger datasets. Whereas Time Fading Apriori suffered extensively when the minsups became too low as Time Fading Apriori was prevented from being able to prune any of its generated candidate itemsets.

7 Conclusions

While more analysis on more varied data sets (ie. density of data) would be required to say anything definitive about the performance of our Time Fading algorithms, the fairly constant runtime of Time Fading VIPER and Time Fading Eclat compared to the Time Fading Apriori that we saw in the experiment we performed was promising. At the very least, it does show that there are scenarios where vertical mining of time time fading data can be more efficient than the more traditional horizontal techniques.

References

1. Shenoy, P., Haritsa, J., Sundarshan, S., Bhalotia, G., Bawa, M., & Shah, D. (n.d.). Turbo-charging vertical mining of large databases.*Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data - SIGMOD '00*.
2. Leung, C., & Jiang, F. (n.d.). Frequent Pattern Mining from Time-Fading Streams of Uncertain Data. *Data Warehousing and Knowledge Discovery Lecture Notes in Computer Science,* 252-264.
3. Javeed Zaki, M., Parthasarathy, S., Ogihara, M., & Li, W. (1997). New Algorithms for Fast Discovery of Association Rules. *Technical Report*.
4. Zaki, M. (n.d.). Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng. IEEE Transactions on Knowledge and Data Engineering,* 372-390.
5. Leung, C. (2015). *Data Stream Mining* [PowerPoint slides].