Mining Sequential Patterns from Probabilistic with Source Level Uncertainty

Venkata Sasidhar Puli

M-Tech, Department of Computer Science and Systems Engineering, Andhra University, Visakhapatnam, Andhra Pradesh

Abstract: ***Sequential Pattern Mining (SPM) is an important data mining problem. Although it is assumed in classical SPM that the data to be mined is deterministic, it is recognized that data obtained from a wide variety of data sources is inherently noisy or uncertain, such as data from sensors or data being collected from the web from different (potentially conflicting) data sources. Probabilistic database is a popular framework for modeling uncertainty. Recently, several data mining and ranking problems have been studied in probabilistic databases. In this work we proposed one of the uncertainty models for spm, namely source level uncertainty which is covered under the framework of probabilistic databases framework. We give a dynamic programming algorithm to compute the source support probability and hence the expected support of a sequence in a source-level uncertain database. We then propose optimizations to speed up the support computation task. Next, we propose probabilistic SPM algorithms based on the candidate generation and pattern growth frameworks for the source-level uncertainty model and the expected support measure. We implement these algorithms and give an empirical evaluation of the probabilistic SPM algorithms and show the scalability of these algorithms under different parameter settings using both real and synthetic datasets. Finally, we demonstrate the effectiveness of the probabilistic SPM framework at extracting meaningful patterns in the presence of noise.***

Keywords: Uncertainty, SPM, Probabilistic database, optimization.

1. Introduction

In data mining finding frequent sequential patterns from data with constrain to time is an crucial milestone in retail marketing through that it easily predicts next affordable item and dumps in to our own interesting list but, the data which to be mined should be anomaly free. (i.e) the data should be certain, accurate and deterministic in nature.

But the data to be mined in pragmatic world ‘s applications are uncertain, incomplete, incorrect, imprecise, inconsistent due to various regions, generally we term it as noisy or dirty. Classical sequential pattern mining works only on clean data in order to achieve that data preprocessing (or) data cleaning phase is applied prior to the actual mining. but, the problem with that is valuable information loss. Consider a scenario where excessive normalization leads to increase in tables likewise excessive data cleansing leads to non-trivial data loss.

Probabilistic database is a popular framework for modeling nondeterministic data. Probabilistic databases are finite set of possible worlds where each possible world having some probability out of which one possible world becomes true possible world. Classical spm with event database the tuples are in the form of <eid,e, σ > where eid represents event-id associated with timestamp, σ is the source for example in a retail transaction recording the retail transaction is the event, the customer is the source, and event-id is the time of event happen. In spm Uncertainty may occur at any part source, event or event-id(timestamp) or perhaps at all three parts also but we pay attention mainly at source that is source level uncertainty(SLU). Such source level uncertainties are:

- A customer (source) purchases some items(event) in a retail store at the time he fills a identity information and during his successive second visit he again fills the identity information with little contradiction to the previous one. Then uncertainty arises at source attribute.
- The logged In person(source) searches for the sequence name APPLE(event) the search term comes with result {(phone company:0.6), (fruit:0.2), (software solutions:o.1)……}.

2. Related work

The uncertainties occurred in sequential pattern mining are modeled in probabilistic databases framework either as tuple level uncertainty or attribute level uncertainty. In tuple level uncertainty, tuples have existential probabilities and in attribute level uncertainty may arise at event or uncertainty may arise at source attribute.

Yang proposed a model based on uncertainty at event. But, it does not considered source to be uncertain it is just like ELU model where event only uncertain, even less flexible to ELU. So, there is a need to implement a model which compromises source level uncertainty also. Computing the probabilistic frequentness of an item set in an uncertain database is a computationally expensive task (sometimes it takes non polynomial time) so approximating the probabilistic frequentness of an item set is implemented.

Dynamic programming based algorithm is implemented for computing expected support and then, extended the classical SPM algorithms based on the candidate generation and pattern growth frameworks to work under probabilistic settings for the SLU model using the expected support measure.

In all, we implemented two candidate generation algorithms, one based on a breadth-first and one based on a depth-first exploration of the search space, as well as a pattern growth algorithm based on the idea of projected database.

Empirical evaluation of the optimizations and algorithms that we implemented are demonstrated by the scalability of our algorithms under different parameter settings using both synthetic and real datasets and also evaluated the effectiveness of the probabilistic SPM framework at extracting meaningful patterns in the presence of noise

3. Methodology

Expected Support

We define the expected support of a sequence s in a probabilistic database Dp using possible world semantics. As every possible world D* is a (deterministic) database, the support of s in D*, denoted by Sup(s,D*). We then define the expected support of a sequence s in a probabilistic database D^p as follows:

$$ES(s, D^P) = \sum_{D^P \in PW(D^P)} \Pr[D^*] * Sup(s, D^*).$$

We now give examples of computing the expected support of a sequence h(a)(b)i for each of the sample probabilistic databases using the possible worlds.

Computing dp in slu database

		$(a, d : 0.6)$	$(a, b : 0.3)$	$(b, c : 0.7)$
	A[0,0] = 1	A[0,1] = 1	A[0,2] = 1	A[0,3] = 1
$\langle(a)\rangle$	A[1,0] = 0	A[1,1] = 0.4 × 0+ 0.6 × 1 = 0.6	A[1,2] = 0.7 × 0.6 + 0.3 × 1 = 0.72	A[1,3] = 0.72
$\langle(a)(b)\rangle$	A[2,0] = 0	A[2,1] = 0	A[2,2] = 0.7 × 0+ 0.3 × 0.6 = 0.18	A[2,3] = 0.3 × 0.18 + 0.7 × 0.72 = 0.558

Algorithm Breadth-First Exploration

```
1.  Input: An SLU database Dp and a support threshold θ.
2.  Output: All sequences s with ES(s,Dp) ≥ θ.
3.  j ← 1
4.  L1 ← ComputeFrequent-1(Dp)
5.  while Lj−1 != ∅ do
6.  Cj ← Join Lj−1 with itself
7.  Prune Cj
8.  for all s ∈ Cj do
9.  Compute ES(s,Dp)
10. end for
11. Lj ← all sequences s ∈ Cj s.t. ES(s,Dp) ≥ θ.
12. j ← j + 1
13. end while
14. Stop and output L1 ∪ . . . ∪ Lj−1
```

The algorithm function is similar to apriori. The length of a sequence is the number of items in it. A sequence having length j is called a j-sequence. The set of candidate j-sequences is called Cj and the set of frequent j-sequences is called Lj . For j = 2 onwards, the input to the j-th phase is the set Lj−1, and the output is the set Lj . In our BFS approach, the algorithm first makes a pass over the SLU database Dp and computes all the frequent 1-sequences using the fast frequent 1-sequence computation . Then, the following steps are performed in any phase j ≥ 2. The set Lj−1 is used to obtain Cj, and while generating Cj, a-priori pruning is applied to Cj and thus, Cj contains only those candidate j-sequences that pass the apriori pruning. In addition to apriori pruning, probabilistic pruning can also be applied to Cj. Then, we perform the support computation for Cj , and the candidate j-sequences having support at least θ is the set of frequent j-sequences Lj . The algorithm stops when no more frequent sequences can be found, or when no more candidate sequences can be generated, and outputs all the frequent sequences.

Candidate Generation: In the j-th phase, we generate the set of candidate j-sequences Cj from the set of frequent (j − 1)-sequences Lj−1. The objective of candidate generation is to generate the smallest possible superset Cj of the set of frequent j-sequences Lj .

Computing Expected Support: The expected support for all the sequences s ∈ Ni,j , and while performing this task, we intend to reuse already computed results in order to save CPU cost.

S-extension: We generate an S-extension t of s by appending an item {x} as a new element to s. We first apply partial apriori pruning to t, say if t is a j-sequence, we first check to see if all the lexicographically smaller (j − 1)-subsequences of t that would have already been explored in the mining process are also frequent.

Further, suppose that we S-extend a sequence s with an item xi ∈ L1 to obtain t = hsi · {xi} and suppose that t is not frequent. extend s with some other item xj ∈ L1 to obtain t′ = hs · {xj}i and t′ is frequent. The apriori property that hs · {xj} · {xi}i can not be frequent either as it contains an infrequent sub-sequence hs·{xi}i. Thus, if an extension of s using xi is not frequent, mark xi as an invalid S-extension for s and do not consider extending any of the sequences for which s is a prefix with xi

I-extension: In I-extension t of s by appending an item {x} to the last element in s. Similar to the S-extension case, and then partial apriori pruning to t. Then, suppose that a sequence s = hs1, . . . , sqi is I-extended with an item xi ∈ L1, i.e. t = hs1, . . . , sq ∪ {xi}i, and suppose that t is not frequent. We mark xi as an invalid I-extension for s and do not consider extending any of the sequences for which s is a prefix with xi. For example, for a sequence h(a)i, suppose that {b} is an invalid I-extension as h(a, b)i is not frequent then, we do not need extending h(a)(c)(a)i
with {b} either.

Depth-First Traversal

```
1.  Input: SLU probabilistic database Dp and a sequence s.
2.  Output: All possible extensions of s with ES(s,Dp) ≥ θ.
3.  function TraverseDFS(s, L1)
4.  L ← ∅
5.  for all valid x ∈ L1 in order do
6.  t ← hs · {x}i {S-extension}
7.  if t is not pruned then
8.  Compute ES(t,Dp)
9.  if ES(t,Dp) ≥ θ then
10. L ← L ∪ t
11. TraverseDFS(t, L1)
```

```
12. end if
13. else
14. Mark {x} as invalid S-extension item.
15. end if
16. t ← hs1, . . . , sq ∪ {x}i {I-extension}
17. if t is not pruned then
18. Compute ES(t,Dp)
19. if ES(t,Dp) ≥ θ then
20. L ← L ∪ t
21. TraverseDFS(t, L1)
22. end if
23. else
24. Mark {x} as invalid I-extension item.
25. end if
26. end for
27. return L
28. end function
```

The algorithm first makes a pass over the SLU database Dp and computes all the frequent 1-sequences L1. Assume that the sequences in L1 are in ascending order. Next, starting with each frequent 1-sequence s ∈ L1, s is both S- and I-extended with every valid item x ∈ L1 and then for all extensions t of s, and then apply partial apriori pruning to t. In addition to partial apriori pruning, probabilistic pruning could also be applied. If t passes the pruning test, we compute the expected support of t in Dp and if t is found to be frequent, we keep exploring the search space by extending t using valid x ∈ L1, recursively. The algorithm stops when no more candidate sequences can been generated or when no more frequent sequences can be found, and outputs all the frequent sequences.

Support Computation:
Given a sequence s, the support computation task is to compute the expected support of s in the probabilistic database Dp. We propose two optimizations for this. 1. Observe that for a sequence t, where t is an S- or I-extension of s, for source i, if Pr[s _ Dpi] = 0, then Pr[t _ Dpi] = 0, i.e. if source i does not support s it cannot support t. When computing the expected support of s in Dp, we keep track of all the sources where Pr[s _ Dpi] > 0, denoted by Ss, and when computing the expected support of t in Dp, we need only to visit the sources in Ss.

2. Further, when doing the support computation for s, we save the Bi,s array with every source i ∈ Ss. When considering an S- or I-extension of s, we can use incremental support computation by reusing results from Bi,s. As the Bi,s arrays are stored for all sources i ∈ Ss with each recursive call of DFS, in the worst case, a source may store up to j arrays, if s is a j-sequence.

Pattern-Growth Algorithm

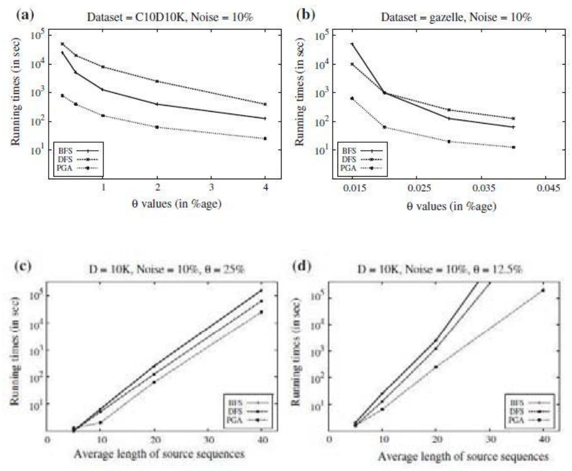
```
1. Input: SLU probabilistic database Dp and support threshold θ.
2. Output: All sequences s with ES(s,Dp) ≥ θ.
3. L1 ← ComputeFrequent-1-sequences(Dp)
4. for all sequences x ∈ L1 do
5. Compute Bi,x arrays
6. Call ProjectedDB(x, Ds,p)
7. end for
8. function ProjectedDB(s, Ds,p)
9. LS ← Compute Frequent S-extensions
10. LI ← Compute Frequent I-extensions
11. Output all Frequent Sequences {s extended with x, for all x in LS and LI}
12. for all x ∈ LS do
13. t ← hs · {x}i {S-extension}
14. Compute Bi,t arrays
15. ProjectedDB(t, Dt,p)
16. end for
17. for all x ∈ LI do
18. t ← hs1, . . . , sq ∪ {x}i {I-extension}
19. Compute Bi,t arrays
20. ProjectedDB(t, Dt,p)
21. end for
22. end function
```

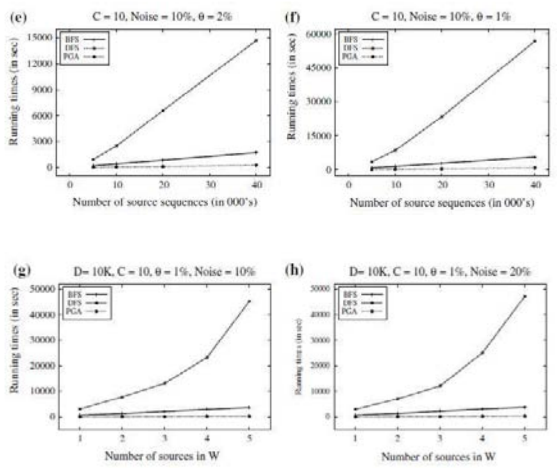
Pattern Growth step:

1. s = hs1, . . . , sqi is a previously discovered frequent sequence.
2. An hsi-projected database Ds,p is available.
3. The Bi,s arrays for all sources i in Ds,p are also available.

Pattern Growth Algorithm

An overview of our pattern-growth algorithm is in Algorithm 5. Assuming that the probabilistic database Dp contains only the frequent items, we first compute the set of frequent 1-sequences L1. Assume L1 is in ascending order. For each 1-sequence hxi, we first create an hxi-projected database Dx,p, and also compute 133 Chapter 6. Probabilistic SPM Algorithms the Bi,x arrays for each source i in Dx,p and then call the ProjectedDB(x, Dx,p) sub-routine recursively. In the ProjectedDB(s, Ds,p) sub-routine, we compute all the frequent S- and I-extensions of s using a modification of fast frequent 1-sequence computation. We call the computation of all S- and I-extensions of a sequence s the pattern-growth step, and elaborate on it in the coming section. Then, for every sequence t which is a frequent S- or I-extension of s, we create a hti-projected database Dt, p and also compute Bi,t arrays, and call the ProjectedDB(t, Dt,p) sub-routine recursively to mine all frequent sequential patterns.





4. Conclusion

The main objective of this paper is to study the sequential pattern mining (spm) problem in probabilistic databases. They are many works on spm in uncertain data. But, these are not capable of presenting basic problems in spm. This paper proposed extensively the concept of dynamic programming for better computing of expected support and two algorithms based on candidate generation and one algorithm based one pattern growth. There are some problems like FP-tree does not produce better results in uncertain FIM and dependencies in several sources is also the problem.

References

[1] Nilesh N. Dalvi, Christopher R´e, and Dan Suciu. Probabilistic databases: diamonds in the dirt. Communications of the ACM, 52(7):86–94, 2009.
[2] O. Hassanzadeh and R. J. Miller. Creating probabilistic databases from duplicated data. The VLDB Journal, 18(5):1141–1166, 2009.
[3] Graham Cormode, Feifei Li, and Ke Yi. Semantics of ranking queries for probabilistic data and expected ranks. In ICDE, pages 305–316. IEEE, 2009. ISBN 978-0-7695-3545-6.
[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms (3rd edition). MIT Press, 2009. ISBN 978-0-262-03384-8.
[5] Aggarwal, C.C., Li, Y.,Wang, J.,Wang, J.: Frequent pattern mining with uncertain data. In: KDD. pp. 29{38 (2009)
[6] Agrawal, R., Srikant, R.: Mining sequential patterns. In: ICDE. pp. 3{14 (1995)