Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 167 (2020) 2429–2436

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Intelligence and Data Science (ICCIDS 2019)

Integrated Probabilistic Data Structure For Accurate and Scalable Sequence Prediction

Soumonos Mukherjee[a], Uddipta Dutta[b], Jit Sarkar[a,b,*], Dr. Rajkumar R[a,b,*,*]

[a]*School of Computer Science and Engineering, Vellore Institute of Technology, Vellore-632014, India*
[b]*School of Electrical Engineering, Vellore Institute of Technology, Vellore-632014, India*
[a,b,*]*Persistent Systems Pvt. Ltd, Nagpur-44002 , India*
[a,b,*,*]*School of Computer Science and Engineering, Vellore Institute of Technology, Vellore-632014, India*

Abstract

The inevitable success of predictive analytics lies in the prospect of cutting edge approaches proposed in the domain of data mining and machine learning. Frequent itemsets, sequential pattern mining has been the major sub domains of the data mining since its inception with human work. Sequence prediction is a rather more targeted approach towards the predictive data mining. Sequential pattern mining approaches, while being very efficient in finding associations and interactions of difference itemsets and discovering hidden pattern in the transactional or graph datasets, sequence prediction is better focused on predicting the next item of an existent sequence. Applications of sequence prediction has been in pattern prediction, stock value prediction, websites prefetching, fraud detection, breach prediction, text prediction, product recommendation in extensive manner. Several state-of-the-art algorithms and models have been proposed starting from graphical models, markovian models and tree based structures have been proposed by data mining. Machine learning models like recurrent neural networks with LSTM blocks work pretty well in terms of performance. Our aim is to propose an integrated data structure with better performance measures (space-time efficiency, scalability) and which can be effectively used in online learning setting-stream processing algorithm. The paper also describes the extension prospect of this algorithm which can be used in the time-series forecast.

Keywords: Predictive analytics; SPM; FIM; Sequence prediction

1. Introduction

Sequence prediction is one of the most cultivated areas of predictive data mining disciplinary. Application of the predictive sequence mining and probabilistic predictive models is seen to be extensive in a range of sectors. The pattern mining or itemset mining algorithms mine the patterns in sequential databases by applying association rule or probabilistic modelling. These have count mechanisms which can be used to predict the sequences but are costly to update in a stream of sequence after each iteration. The solutions given towards are either lossy or not space-efficient. Markovian models like Dependency graph or partial pattern matching approaches not universal. Modern approaches based on markovian principle try to make the models more versatile but are very much time inefficient. Arrival of neural networks have considerably improved the dependency constraint of markovian networks but are lossy and therefore not very efficient to be implemented iteratively in the vast data streams. All the setbacks indicate towards a weakness for scalable and accurate implementation over data stream. Our model addresses both the insufficiencies to overcome the pitfalls of the prediction application.

1.1. Problem Statement

The problem statement of our particular research is that, given a stream dataset of sequence $D=\{S_1, S_2, S_3,....S_n\}$ as a training sequence, where n is an incremental variable. Each Sequence (S_n) is a list of items as $S_n=\{X_1, X_2, X_3.....X_m\}$ where m is an integer, $m>>100$ in real-world use cases. And items (X_n) are chosen from a Universal set U_x. Therefore, $U_x=\{X_1, X_2, X_3.....X_m X_{m+1},..X_h\}$ and $h>m$ we have to build a scalable prediction model T_{pr} which will predict the next items of the sequence $S_n(X_{n+1}, X_{n+2}..)$.

2. Existing models

Since the inception of predictive data mining, a huge number of research has been going on in this field[14,15].We have mentioned the major ones. The first model which was proposed based on markovian principle was Prediction by Partial Matching (PPM)[1] where prefix subsequences are linked to their suffix counterparts with outgoing arcs and transition probabilities attached to each. Upcoming suffix of a given sequence is predicted by matching the lask k items with one of the nodes in the graph. This model has a problem with handling variants of subsequences and loses accuracy as variations happen in the targeted subsequences as for a given k only k-th markovian predictor is used. The enhancement to this algorithm came up with All-k-order Markov[2] where all the markovian predictors from 1 to k are inclusively used. But computing k markovian predictor is costly both in terms of time and storage. Dependency graph[3] aims to be more universal introducing approximations to the graphical predictor models. For a crafted attribute x, it connects prefix and suffix nodes with dependency arcs only if the suffix occurs after the particular prefix subsequence within last x items. For given node B and A, the weight of the arc between them has a dependency weight of the conditional probability $[\ P(B|A)/P(A)\ ]$ between them till a look up window of value x. Sequential rule mining[4,5,6] and Sequential pattern mining[7,8,9] algorithms use association rule mining, pattern growth and other vertical approach to predict the suffix sequences. Neural network models[10,11] and deep belief networks[12] predict the sequence and patterns with unsupervised or semi supervised learning approach using transformations and rule selection approach. A common drawback that all of the above described models share is that those are not lossless and cannot handle noisy data, needs huge preprocessing. Compact prediction tree[13] model is built to be a lossless model for near accurate prediction of suffix sequences by using a 'trie' based structure and a hash indexing method on hash maps. The major problem of this model is that, it is not feasible in terms of its spatial complexity and due to the recomputing principle of its paradigm, can not be used scalably in stream data settings. Dimensionality of the data is again a big problem for machine learning algorithms to work on and it poses a set back in the need of scalable implementation of the above algorithms.

3. Proposed model

3.1. Design of model

Our model proposes a tree based structure which is space and time efficient, almost lossless and will increase the prediction accuracy to a considerable amount from the existing systems. This consists of 3 sub modules. The description of each module is given below.

3.1.1. Module-1: Sequence Tree

This is a tree data structure where each node contains 3 attributes. An item from the training sequence, a list containing the children nodes and a pointer towards its parent node. While training of the algorithm, sequences are taken as input one by one and the item gets appended into the tree nodes recursively. Whenever a sequence is entered, a look up is performed to search the availability of the items of the sequence. If the first item is present it moves on to check the next item. When it encounters an item that is not present, the new item is stored in a child node of the immediately preceding parent node and the pointer moves to the child node to be directed towards the parent. The process is performed iteratively to store the sequences. This way, the sequences are stored as branches of the tree.

Analysis

Time complexity for linear insertion is O(l) where l is the sequence length. Space complexity for the worst case scenario is O(nl$_{a)}$where *la* is the average length of sequences. Though, due to repetitive overlaps between sequences, the actual complexity is lesser than the expression. Moreover, the optimizations applied on the structure improves the complexity by magnitudes.

3.1.1.1. Optimization and compression of the Sequence tree

3.1.1.1.1. Frequent Itemgroup replacement

The repetitive and frequent substrings (items that occur together with repetition in many sequences) are identified and replaced by sequential pattern mining algorithm. Pattern growth algorithm have been used for completing the task. To give an example, if 3 sequences (*S1: a b b c d, S2: a c b c b and S3: a d a c b c c a*) are considered, it can be seen that a subgroup (*b c*) is common to all of them. If the item-lists are appended as it is, in very large datasets it sometimes pose to huge space usage for these repetitive groups. A unique identifier that is not present as any item in the dataset can replace these groups (e.g: y[b c]).

3.1.1.1.2. Nodal Compression

After the sequence tree is built, the tree is traversed with a bottom up approach by looking up to each branch from the look up table (the last stored item). There are always some of the branches which have single child node for each parent node. These branches are compressed to be represented as a single node containing all the items in the correct order. This step preferably should be performed after the training of the algorithm is over. Dynamic implementation of this may pose to prediction error and reconstruction failure.

3.1.2. Module-2: Probabilistic filter Index

The second structure is a scalable bloom filter. Bloom filter is an approximate data structure which does not store the exact items but stores a probabilistic in the allocated spaces for storing the sketch of the items of the sequence with key value pairs. An existing model uses hash table to store the items of the sequence for indexing. Hash table sometimes poses very less efficient status of space complexity if the sequence sizes are big and if the setting is dealing with dense datasets. Bloom filter is a probabilistic enhancement of the hash table and with an ensured zero error probability in false negatives. However, bloom filter can produce false positive and to reduce the false positive probability, some optimization measures have been imposed. Scalable bloom filter outperforms conventional bloom filters in stream data input setting. Bloom filter works at its best when fixed amount of bits have been fixed to store the entire data. Scalable bloom filter allow the filter bits to grow as a function of false positive probability and size. Traditional bloom filter needs to determine the total number of elements to be able to calculate optimal values of *M* (total size of the bit arrays) and *k*(number of required hash functions). Scalable bloom filter efficiently addressed this limitation.

3.1.2.1. Modifications and optimizations

3.1.2.1.1. Filter split

This starts with a traditional bloom filter and partitions the total *M* bits into *k* parts and initiate *k* hash functions. This operation produces k slices of bit array with size *m=M/k* . Each element is therefore expressed with k bits. This removes the bias of false positive considerably.

3.1.2.1.2. Optimal Set size and Error probability

In traditional bloom filter, it can be derived that if Error probability is *P* and *p* is the fill ratio and *n* being the number of elements to be inserted, then

$$p = 1 - (1 - 1/l) \wedge n, \quad \text{for } P = p \wedge k \qquad \text{(i)}$$

Applying the Taylor series approximation on this equation we get,

$$p \approx 1 - e\wedge\,(-n/l) \qquad \text{(ii)}$$
$$n \approx - m\, ln(1 - p) \qquad \text{(iii)}$$
$$m = M \times [ln\, p\; ln(1 - p)] \,/\, (-\,ln\, p) \;\; \text{for } M{=}mk \qquad \text{(iv)}$$

It infers that for any value of *P* and *M*, *n* is maximized at *p*=½. So, the optimal filter performance is achieved when the slices are half filled to their capacity. From equation (iii) it can be said that:

$$n = M \times (ln\, 2) \wedge 2/\; |ln\, P| \qquad \text{(v)}$$
$$\text{and } k = log_2\, 1/p \qquad \text{(vi)}$$

With these bounding approximations, we are able to determine the optimal values of *k, m* and *n* defining the value of the false positive error probability as *P*.

3.1.2.2. Scalable approach

The scalable bloom filter is made up of multiple traditional bloom filters. In typically an input data stream, when the existing filter is filled according to the limit by fill-ratio, another one is added to the index. Each subsequent filter is initialized to a tighter upper bound of false positive probability. The whole setting thus converges well even in an infinite stream of sequences. The tightening ratio is introduced as *r* and set its value at a range of .7 to .9 to obtain the maximized performance. While choosing the size of the successive filters it is calculated orthogonally to the required false positive probability. Let the filters grow in a scalable manner with exponential expansion ($m, ms, ms^2 \ldots. ms^{T-1}$) with essentially keeping value of s as a power of 2 (preferably 2 for small and 4 for large growth possibilities).

3.1.2.3. Index-Counter

As the aim is to develop this prediction algorithm particularly for online setting, in a data stream, the sequence count cannot be approximated from the initial phase of training. This poses a hindrance to declare the initial size of the bit array index. The solution is to maintain an incremental counter which is initialized at 0 and adds up to the value at which the number of items has been inserted. This is a space efficient approach and takes only $O(n)$, n is a constant value, in fact, $n<<10$ in real-world scenario.

3.1.2.4. Optimal Hash function

After rigorous experimental evaluation of various hash functions SDBM hash function is found to be the best fit for our case scenarios. It works well in the strings and sequences or alphanumeric variables. It offers a good distribution of keys with fewer splits and works extremely well in the datasets with high variance. The general form is described as $\text{hash}(i) = \text{hash}(i - 1) * k + \text{str}[i]$, where k is a constant, that can be picked up on a trial and error manner with experimental outcomes. A Duff's device version of SDBM hash function is claimed to be the fastest in terms of computational time.

Analysis

The scalable growth of the multiple successive filters and the declared size of the bit arrays are defined by finding an optimal k and m value. A hash table with a fixed size $[U]$ and n items inserted to it will have a space complexity of $O(n\log[U])$. In practical implementation hash tables take up a space of approximately $O(nw+bw)$ complexity where n is the number of sequences, w is the number of unique elements and b is the size of an element. An optimal m which is dynamically implemented with determining the input stream an optimal value of s is set to the lowest possible power of 2 and it results in a $m<<[U]$ and the space complexity is $O(m)$. For an optimal k, the time complexity for insertion and search is $O(k)$. It has been observed that scalable implementation of multiple sliced filters improves on the total space usage as compared to linear implementation of one large or multiple small bloom filters.

3.1.3. Module-3: Look up table

An incremental array structure storing the last items of the sequences. It points towards the last node of a branch (sequence) in the sequence tree. This integration of an accurate data structure gives the algorithm robustness and increases the accuracy which stores actual items and which can be traced back to restore the appended dataset when needed. It improves the efficiency without sacrificing much space. It also replaces the exhausting runtime computation as it updates with the input of each sequence while training.

Analysis

This part of the algorithm takes up $(nb+np)$ space where n is the number of sequences, b is the size of each item and p is the size of a pointer.

4. Training

The training is done by a stream input paradigm where datasets are essentially transactional database input with sequences containing items, preferably denoted by unique words or numbers which can be transformed by the real-world transaction data. The training process of the algorithm takes an overall of $O(n)$ time. The optimizations and the compression are performed consecutively along with the training data input. Dataset properties have tangible effects on the compression success.

4.1. Score-table

A score counting mechanism is introduced to calculate prediction score while the prediction occurs. A score table is an array which stores a count for every discovered precedent item of the search tree. The score calculation mechanism is described in the prediction phase.

5. Prediction

A user defined variable is asked as input while predicting the suffix subsequences of a given sequence. Let *c* be the prefix length. For a given sequence *S*, First step is to find all sequences with higher likelihood within the training storage. To find the maximum likely sequences one can use bitwise AND operation for intersection of the bit-arrays or Locality sensitive hashing can be used on the probabilistic index. Then the resulting bit arrays are searched as to search for sequences by using the lookup table by a trace back reconstruction mechanism. The algorithm captures all the consequent suffix subsequences for all the reconstructed branches and performs a count operation with the support value of the suffix items. The support counts are defined as the total number of occurrences of the suffix items in the targeted sequences. The ties between items with same support value are prioritized with a higher confidence value which implies the support value divided by the total number of items in the reconstructed tree. The count scores are stored in a hash table and compared. The item with the maximum count score gives the output as the predicted item.
The search operation takes $O(k)$ time on the index for k hash functions and takes $O(n)$ time for reconstruction of the tree.

6. Performance Evaluation

Our proprietary algorithm is implemented alongside implementing the other trivial and state-of-the-art algorithms. This algorithm particularly performs at its best for the datasets consisting of very large sequence count. The improved method of indexing takes up much less space than the previous methods. This algorithm has outperformed the existing models in several aspects. The results are transcribed below. The project has used an intel core i7 gen-4 processor with 4 GB of available RAM for the implementation purpose of all the existing models and our proposed approach.

6.1. Dataset description

The dataset is a natural language text on religious book of Christian named ‘BIBLE’. It has got 76 distinct item counts and 32,529 sequence count with every sequence being of an average of 131 items in length. Average item occurrence per sequence is approximately 5.

6.2. Performance measure

4 performance evaluation parameters are used: Accuracy, Space efficiency, Time-efficiency and Scalability.

Table 1. Primary performance measures comparison

Measure	Proposed Model	CPT	PPM	AKOM	DG
Accuracy	82.24	82.26	33.36	79.22	10.12
Size (nodes)	1015	11124	105	72727	83
Training-time (seconds)	188	176	556	834	5051
Testing-time	56	177	221	78	185

Table 2. Scalability (seq. count VS. space requirement)

	Space Exhausted (Nodes)				
Sequence	Proposed Model	CPT	PPM	AKOM	DG
2000	8400	10501	2102	110502	1804
5000	12553	21005	2780	123071	2607
10000	18550	32004	4404	323064	5676
20000	22523	63232	8251	515055	9550
30000	30523	94565	12541	772313	15231

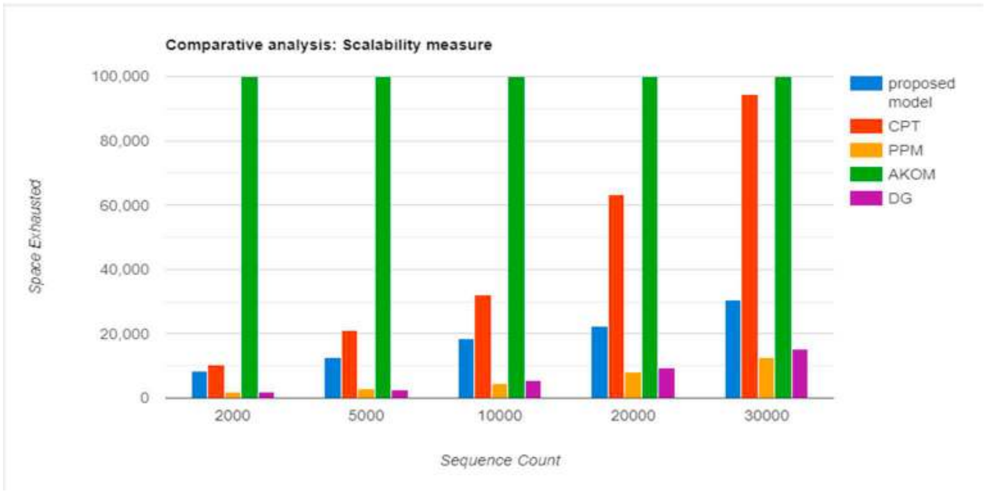

Fig. 1. Comparative analysis: Scalability measure (**N.B**: AKOM runs out of memory from the first iteration)

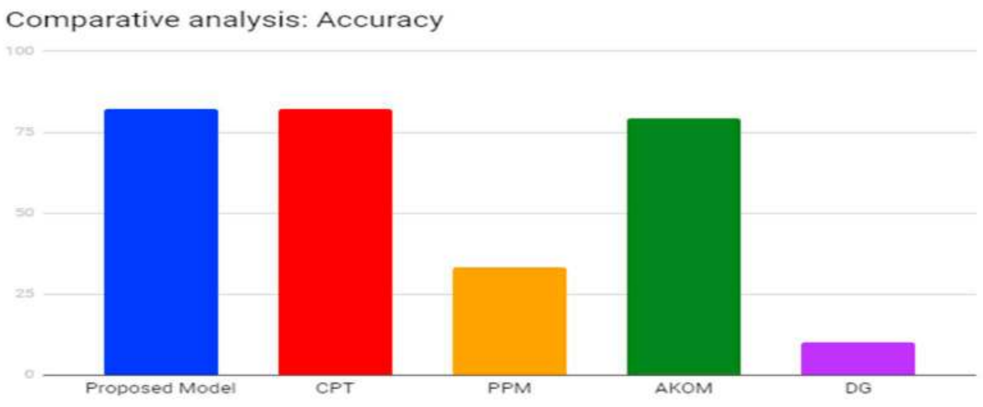

Fig. 2. Comparative analysis: measured accuracy

-From these experimental comparison, it can be observed that our proposed model gives output of the predicted subsequences at the same level of accuracy as compact prediction tree algorithm and slightly better than All-K-order Markov method. However, the approach exhausts far less nodes than CPT and AKOM. The training and testing time are also approximately analogous to those of CPT. Coming to the evaluation of scalability of algorithms, The algorithm shows a linear growth over the incremental number of sequences, the memory exhaustion scales up and becomes synchronous to the number of sequences. The traditional model like AKOM has a huge spatial requirement which is unrealistic to afford in the real-time settings. CPT also claims much memory than our proposed approach. CPT can be implemented as a Batch-processing algorithm but can not be used in data streams for its iterative and recomputing manner of progression. Our model works perfect for very large datasets and for long sequence lengths in online input setting.

7. Conclusion

The paper presents a model for efficiently predicting the suffix subsequences for a given sequence. The model is trained over a stream of data with incremental number of sequences. The algorithm can be implemented in a scalable manner in big data setting and will outperform the state-of-the art existing algorithms in time and space complexity by magnitude amount. The further research on this model will be optimizing the hash functions of the index filters to generate results with higher accuracy and compressed optimization of the tree structures and nodes for better spatial efficiency. The tree search methods and reconstruction are also subjected to enhancements. In the continuing research, we aim to propose the extended version of this algorithm to efficient time-series forecast.

References

1. Cleary, J., Witten, I.: Data compression using adaptive coding and partial string matching. IEEE Trans. on Inform. Theory, vol. 24, no. 4, pp. 413-421 (1984).
2. Pitkow, J., Pirolli, P.: Mining longest repeating subsequence to predict world wide web surng. In: Proc. 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, CO, pp. 13-25 (1999).
3. Padmanabhan, V.N., Mogul, J.C.: Using Prefetching to Improve World Wide Web Latency, Computer Communications, vol. 16, pp. 358-368 (1998).
4. Sun, R., Giles, C. L.: Sequence Learning: From Recognition and Prediction to Sequential Decision Making. IEEE Intelligent Systems, vol. 16 no. 4, pp. 67-70 (2001).
5. Fournier-Viger, P., Gueniche, T., Tseng, V.S.: Using Partially-Ordered Sequential Rules for Sequence Prediction. In: Proc. 8th Intern. Conf. on Advanced Data Mining and Applications, Springer LNAI 7713, pp. 431-442 (2012).
6. SPADE: An Efficient Algorithm for Mining Frequent Sequences: Machine Learning, 42, 31–60, 2001,Kluwer Academic Publishers (2001).
7. A. Gomariz, M. Campos, R. Marin, and B. Goethals, "ClaSP: An efficient algorithm for mining frequent closed sequences," The Pacific-Asia Conference on Knowledge Discovery and Data Mining,pp. 50–61 (2013).
8. R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," The International Conference on Very Large Databases, pp. 487–499 (1994).
9. K. Gouda, M. Hassaan, and M. J. Zaki, "Prism: An effective approach for frequent sequence mining via prime-block encoding," Journal of Computer and System Sciences, vol. 76(1), pp. 88–102 (2010).
10. Maha Elbayad, Laurent Besacier, Jakob Verbeek, "Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction"Proceedings of the 22nd Conference on Computational Natural Language Learning (CoNLL 2018), pages 97–107 Brussels, Belgium, October 31 - November 1, 2018. c 2018 Association for Computational Linguistics.
11. Yanpeng Zhao et al, "Sequence Prediction Using Neural Network Classifiers", JMLR: Workshop and Conference Proceedings 57:164–169 (2016).
12. Hongda Bu et al, "A new method for enhancer prediction based on deep belief network", BMC Bioinformatics ; 18(Suppl 12): 418 (2017).
13. Philippe Fournier-Viger et al, "Compact Prediction Tree: A Lossless Model for Accurate Sequence Prediction" (2013).
14. Papapetrou, P., Kollios, G., Sclaroff, S., Gunopulos, D.: Discovering Frequent Arrangements of Temporal Intervals. In: Proc. of the 5th IEEE International Conference on Data Mining, pp. 354-361 (2005).
15. Willems, F., Shtarkov, Y., Tjalkens, T.: The context-tree weighting method: Basic properties. IEEE Trans. on Information Theory, vol. 31, no. 3, pp. 653-664 (1995).