Compact Prediction Tree: A Lossless Model for Accurate Sequence Prediction

Ted Gueniche[1], Philippe Fournier-Viger[1], and Vincent S. Tseng[2]

[1] Dept. of Computer Science, University of Moncton, Canada
[2] Dept. of Computer Science and Inf. Eng., National Cheng Kung University, Taiwan
{etg8697, philippe.fournier-viger}@umoncton.ca, tsengsm@mail.ncku.edu.tw

Abstract. Predicting the next item of a sequence over a finite alphabet has important applications in many domains. In this paper, we present a novel prediction model named CPT (*C*ompact *P*rediction *T*ree) which losslessly compress the training data so that all relevant information is available for each prediction. Our approach is incremental, offers a low time complexity for its training phase and is easily adaptable for different applications and contexts. We compared the performance of CPT with state of the art techniques, namely PPM (*P*rediction by *P*artial *M*atching), DG (*D*ependency *G*raph) and All-K-th-Order Markov. Results show that CPT yield higher accuracy on most datasets (up to 12% more than the second best approach), has better training time than DG and PPM, and is considerably smaller than All-K-th-Order Markov.

Keywords: sequence prediction, next item prediction, accuracy, compression

1 Introduction

Given a set of training sequences, the problem of sequence prediction consists in finding the next element of a target sequence by only observing its previous items. The number of applications associated with this problem is extensive. It includes applications such as web page prefetching [3, 5], consumer product recommendation, weather forecasting and stock market prediction.

The literature on this subject is extensive and there are many different approaches[6]. Two of the most popular are PPM (*P*rediction by *P*artial *M*atching)[2] and DG (*D*ependency *G*raph) [5]. Over the years, these models have been greatly improved in terms of time or memory efficiency [3, 12] but their performance remain more or less the same in terms of prediction accuracy. Markov Chains are also widely used for sequence prediction. However, they assume that sequences are Markovian. Other approaches exist such as neural networks and association rules [9]. But all these approaches build prediction lossy models from training sequences. Therefore, they do not use all the information available in training sequences for making predictions.

In this paper, we propose a novel approach for sequence prediction that use the whole information from training sequences to perform predictions. The hypothesis is that it would increase prediction accuracy. There are however several important challenges to build such an approach. First, it requires a structure for storing the whole information efficiently in terms of storage space. Second, the structure should be efficiently updatable if new sequences are added. Third, it is necessary to define an algorithm for performing predictions using the data structure that is time efficient and generate accurate predictions.

We address all these challenges. First, we propose an efficient trie-based data structure named CPT (Compact Prediction Tree) which losslessly compress all training sequences. The construction process of the CPT structure is incremental, offers a low time complexity and is reversible (i.e. it is possible to restore the original dataset from a CPT). Second, we propose an efficient algorithm to perform sequence predictions using the CPT structure. Thanks to CPT's indexing mechanism, the algorithm can quickly collect relevant information for making a prediction. Third, we introduce two strategies that respectively reduce the size of CPT and increase prediction accuracy. Lastly, we perform an extensive experimental study to compare the performance of our approach with state of the art sequence prediction algorithms, namely PPM [2](Prediction by Partial Matching), DG [5] (Dependency Graph) and All-Kth-Order Markov [8], on several real-life datasets. Results show that CPT yield superior accuracy in most cases.

This paper is organized as follows. In section 2, we formally present the prediction problem and discuss related work. In section 3, we present CPT, explain how its substructures are built and how it is used to perform predictions. In section 4, we describe an experimental study. Finally, in section 5, we present our conclusions.

2 Preliminaries and Related work

Problem definition. Given a finite alphabet $I = \{i_1, i_2, ..., i_m\}$, an individual sequence is defined as $S = \langle s_1, s_2, ..., s_n \rangle$, a list of ordered items where $s_i \in I$ $(1 \leq i \leq m)$. Let $T = \{s_1, s_2, ..., s_t\}$ be a set of training sequences used to build a prediction model M. The problem of sequence prediction consists in predicting the next item s_{n+1} of a given sequence $\langle s_1, s_2, ..., s_n \rangle$ by using the prediction model M.

Related work. *Prediction by Partial Matching* [2] (PPM) makes predictions based on the last K items of a sequence, where K defines the order of the model. A PPM model can be represented as a graph where prefix subsequences are linked to suffix subsequences by outgoing arcs having transition probabilities. In a K-Order PPM, the suffix of a given sequence is predicted by matching its last k items with one of the node. This approach has been proven to yield good results in certain areas [2, 3]. However, an important drawback is its rigidness toward patterns that it can learn. The smallest variation in a subsequence will affect the prediction outcome, and thus prediction accuracy. This problem become worse for noisy datasets. In a K-Order PPM model only the Kth-order Markov predictor is used. In the *All-K-Order Markov Model* [8], all Markov predictors from 1 to K inclusively are used. This has the advantage of yielding higher accuracy in most case [3]. But it suffers from a much higher state and space complexity. A lot of research has been done to improve the speed and memory requirement of these approaches, for example by pruning states [3, 12, 6].

The *Dependency Graph* (DG) [5] model is a graph where each node represents an item $i \in I$. A directionnal arc connects a node A to a node B if and only if B appears within x items from A in training sequences, where x is the *lookahead window length*. The weight of the arc is $P(B|A)/P(A)$.

There are many other approaches to sequence prediction such as using sequential rules [4], neural networks and Context Tree Weighting [10] (see [9] for an overview). However, all these approaches build lossy models, which may thus ignore relevant information from training sequences when making predictions. In this work, we propose a lossless prediction model. Our hypothesis is that using all the relevant information from training sequences to make predictions would increase prediction accuracy.

3 The Compact Prediction Tree

In this section, we present our approach. It consists of two phases: training and prediction.

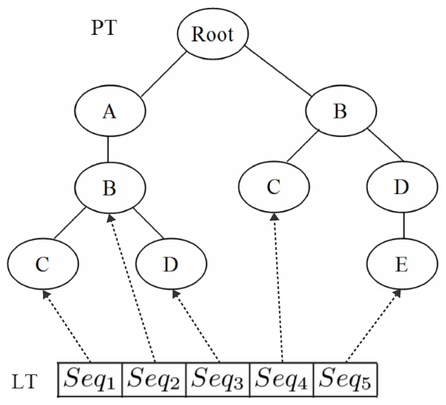
3.1 Training

In the training phase, our prediction model named the Compact Prediction Tree is built. It is composed of three data structures: (1) a Prediction Tree (PT), (2) an Inverted Index (II) and (3) a Lookup Table (LT). The training is done using a training dataset composed of a set of sequences. Sequences are inserted one at a time in the PT and the II. For example, Figure 1 show the PT, II and LT constructed from sequences $\langle A, B, C\rangle$, $\langle A, B\rangle$, $\langle A, B, D\rangle$, $\langle B, C\rangle$, $\langle B, D, E\rangle$.

The *Prediction Tree* is recursively defined as a node. A node contains an item, a list of children nodes and a pointer to its parent node. A sequence is represented within the tree as a full branch or a partial branch; starting from a direct child of the root node. The prediction tree is constructed as follows: given a training sequence, we check if the current node (the root) has a direct child matching the first item of this sequence. If it does not, a new child is inserted to the root node with this item's value. Then, the cursor is moved to the newly created child and this process is repeated for the next item in the training sequence. The construction of this tree for N training sequences takes $O(N)$ in time and is done by reading the sequences one by one with a single pass over the data. The space complexity of the PT is in the worst case $O(N * averageLengthOfSequences)$ but in the average case the PT is more compact because the branches often overlap by sharing nodes. Two sequences share their first v nodes in the PT if they share a prefix of v items. The PT is incrementally updatable and is fast to construct.

The second structure is the *Inverted Index*. It is designed to quickly find in which sequences a given item appears. Hence, it can also be used to find all the sequences containing a set of items. The II is defined as a hash table containing a key for each unique item encountered during the training. Each key leads to a bitset that indicates IDs of the sequences where the item appears. A bitset contains n bits, where n is the number of training sequences. The presence of an item in the s-th sequence is indicated by setting the s-th bit to 1 in its bitset, and 0 otherwise. The II, just like the PT, has an average construction time of $O(n)$ and takes $((n + b) * u)$ bytes where n is the number of training sequences, u is the number of unique items and b is the size of an item in bytes.

The third and last structure is the *Lookup Table*. It links the II to the PT. For each sequence ID, the LT points to the last node of the sequence in the PT. The LT purpose is to provide an efficient way to retrieve sequences from the PT using their sequence IDs. The LT is updated after each sequence insertion in the PT. Its time complexity is $O(n)$ where n is the number of sequences. In terms of size, this data structure takes $n * (b + p)$ bytes where n is the number of sequences, b is the size of an item in bytes and p is the size of a pointer in bytes. The addition of the LT to the PT makes it a lossless representation of the training set of sequences, i.e. it allows restoring the original dataset.



II

Item	Seq_1	Seq_2	Seq_3	Seq_4	Seq_5
A	1	1	1	0	0
B	1	1	1	1	1
C	1	0	0	1	0
D	0	0	1	0	1
E	0	0	0	0	1

Fig. 1. A Prediction Tree (PT), Inverted Index (II) and Lookup Table (LT)

The training process is really fast ($O(n)$). The CPT take more or less space depending on the dataset. If many sequences share common prefixes, a greater compression is achieved. Note that the PTitself could be further compressed by replacing frequent subsequences by single nodes or pruning infrequent nodes. These optimizations are outside the scope of this paper and will be investigated in future work.

3.2 Prediction

In the prediction phase, our prediction model is used to perform predictions. Let x be an integer named the prefix length. Making a prediction for a given sequence S is done by finding all sequences that contains the last x items from S in any order and in any position. We call these sequences the *sequences similar to* S and they are used to predict the next item of S. The process of finding the sequences similar to S is implemented efficiently by using the II. It is done by performing the intersection of the bitsets of the last x items from S. The resulting bitset indicates the set of sequences similar to S. Using the LT, it is trivial to access these sequences in the PT. For each similar sequence Y, the algorithm capture its consequent w.r.t S. The *consequent of a sequence* Y *with respect to a sequence* S is the subsequence of Y starting after the last item in common with S until the end of Y. Each item of each of those consequents are then stored in a structure named *Count Table* (CT). A CT is defined as a hash table with items as keys and a score as associated value. This structure holds a list of possible candidate items and their respective score for a specific prediction and hence is unique for each individual prediction task. The item with the highest score within the CT is the predicted item. The primary scoring measure is the support. But in the case where the support of two items is equal, the confidence is used. We define the *support of an item* s_i as the number of times s_i appears in sequences similar to S, where S is the sequence to predict. The *confidence of an item* s_i is defined as the support of s_i divided by the total number of training sequences that contain s_i (the cardinality of the bitset of s_i in the II). We picked the support as our main scoring measure because it outperformed other measures in terms of accuracy in our experiments.

Performing a prediction is fairly fast. The time complexity is calculated as follows. The search for similar sequences is performed by bitset intersections (the bitwise AND operation), which is $O(1)$. The construction of the CT is $O(n)$ where n is the number of items in all consequents. Finally, choosing the best scoring item is done in $O(m)$ where m is the number of unique items in all consequents. In terms of spatial complexity, the CT is the only constructed structure in the prediction process and its hashtable only has m keys.

3.3 Optimizations

Sequence Splitter. The first optimization is done during the training phase while the PT is being constructed. Let *splitLength* be the maximum allowed length for a sequence. For each sequence longer than *splitLength* items, only the subsequence formed by its last *splitLength* items are inserted in the PT. By using this optimization, the resulting CPT is no longer lossless since sequence information is discarded. Splitting long sequences has for goal to reduce the PT size by reducing the number of possible branches and by enforcing an upper bound on the depth of branches. Intuitively, it may seems that this optimization would negatively affect the prediction's accuracy. But we have observed that it boosts the accuracy by forcing prediction to focus on the latest W items of each training sequence. We also observed that this optimization greatly reduces the prediction time and the CPT size (cf. section 4.3).

Recursive Divider. One of the problem we experienced early in our research is the low coverage of our approach for prediction. Since our model is based on finding similar sequences that share a fixed subset of items T, if some noise is introduced in T, CPT is only able to find similar sequences containing the same noise. To make our approach more flexible, we introduce a recursive method named the Recursive Divider that tries removing the noise from T when searching for similar sequences. This approach works by levels $k = 1, 2...$ *maxLevel*, where *maxLevel* is a constant indicating the maximum number of levels to explore. At level k, for each subset $Q \subset T$ such that $|Q| = k$, the Recursive Divider uses the similar sequences to T/Q to update the CT. Note that each training sequence is only used once for each level to update the CT. If a prediction cannot be made at level k, the Recursive Divider moves to level $k + 1$ if $k + 1 < maxLevel$. In the experimentation section, we show that this technique boosts the coverage of CPT.

4 Experimental evaluation

To evaluate the performance of the proposed prediction model, we performed a set of experiments. Our test environment is made of an Intel i5 third generation processor with 4.5 GB of available RAM on a 64-bit version of Windows8.

4.1 Datasets

We used five real-life datasets representing various types of data. Table 1 summarizes their characteristics. For each dataset, sequences containing less than 3 items were discarded .

BMS is a popular dataset in the field of association rule mining made available for KDD CUP 2000 [11]. It contains web sessions from an e-commerce website, encoded as sequences of integers, representing web pages.

FIFA contains web sessions recorded on the 1998 FIFA World Cup Web site and holds 1,352,804,07 web page requests [1]. Originally, the dataset is a set of individual requests containing metadata (e.g. client id and time). We converted requests into sequences by grouping requests by users and splitting a sequence if there was a delay of more than an hour between two requests. Our final dataset is a random sample from the original dataset.

SIGN is a dense dataset with long sequences, containing 730 sequences of sign-language utterances transcripted from videos [7].

KOSARAK is a dataset containing web sessions from a Hungarian news portal available at `http://fimi.ua.ac.be/data`. It is the largest dataset used in our experimental evaluation.

BIBLE is the religious Christian set of books used in plain text as a flow of sentences. The prediction task consists in predicting the next character in a given sequence of characters. The book is split in sentences where each sentence is a sequence. This dataset is interesting since it has a small alphabet with only 75 distinct characters and it is based on natural language.

Table 1. Dataset characteristics

Dataset	**Sequence count**	**Unique items**	**Avg sequence length**	**Avg item occurence count per sequence**
BMS	15,806	495	6.01	1.00
FIFA	28,978	3,301	32.11	1.04
SIGN	730	267	93.00	1.79
KOSARAK	638,811	39,998	11.64	1.00
BIBLE	32,529	76	130.96	4.78

4.2 Evaluation Framework

We designed a framework to compare our approach with state-of-the-art approaches on all these datasets. The framework is publicly available at `http://goo.gl/hDtdt` and is developed in Java. The following paragraphs describes the evaluation process of our framework.

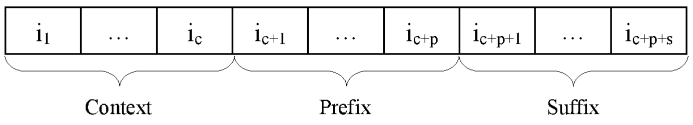


Fig. 2. Sequence splitting (context, prefix, suffix)

Each dataset is read in memory. Sequences containing less than three items are discarded. The dataset is then split into a training set and a testing set, using the 10-fold cross-validation technique. For each fold, the training set is used to train each predictor. Once the predictors have been trained, each sequence of the testing set is split into three parts; the *context*, the *prefix* and the *suffix* as shown in Fig. 2. The prefix and suffix size are determined by two parameters named *PrefixSize* (p) and *SuffixSize* (s). The context (c) is the remaining part of the sequence and is discarded. For each test sequence, each predictor accepts the prefix as input and makes a prediction. A prediction has three possible outcomes. The prediction is a *success* if the generated candidate appears in the suffix of the test sequence. The prediction is a *no match* if the predictor is unable to perform a prediction. Otherwise it is a *failure*. We define three measures to assess a predictor overall performance. *Local Accuracy* (eq. 1) is the ratio of successful predictions against the number of failed predictions.

$$Local\_Accuracy = |successes|/(|successes| + |failures|) \quad (1)$$

Coverage (eq. 2) is the ratio of sequence without prediction against the total number of test sequences.

$$Coverage = |no\_matches|/|sequences| \quad (2)$$

Accuracy (eq. 3) is our main measure to evaluates the accuracy of a given predictor. It is the number of successful prediction against the total number of test sequences.

$$Accuracy = |successes|/|sequences| \tag{3}$$

The above measures are used in our experiments as well as the spatial size (in nodes), the training time (in seconds) and the testing time (in seconds). The spatial size is calculated in nodes because the spatial complexity of all predictors can be represented in terms of nodes. This measure is meant to show the spatial complexity and is not used to determine the exact size of a model.

4.3 Experiments

Overall performance. The goal of the first experiment consists in getting an overview of the performances (accuracy, training and testing time and space) of CPT against DG, 1st order PPM and All-Kth-Order Markov (AKOM). DG and AKOM were respectively tuned with a lookahead window of 4 and with an order of 5, since these values gave the best performance and are typically good values for these algorithms [3, 5, 8]. Results are shown in Tables 2 and 3. Results show that CPT yield a higher accuracy for all but one dataset. DG and PPM perform well in some situations but CPT is more consistent across all datasets. The training time, just like the testing time can be critical for some applications. In this experiment, CPT is always faster to train than DG and All-Kth-Order Markov by at least a factor of 3, and has comparable training time to PPM. The downside of CPT is that making a prediction can take longer than other methods. This characteristic is a trade off for the higher accuracy and is mainly caused by the Recursive Divider optimization described in section 3.3. The coverage is not presented because a high coverage (> 95%) is achieved by all the predictor for all datasets and it is also indirectly included in the *overall accuracy* measure.

Table 2. Comparison of accuracy and model size

Dataset	Overall Accuracy				Size (nodes)			
	DG	CPT	PPM	AKOM	DG	CPT	PPM	AKOM
BMS	36.07	**38.45**	31.12	30.81	**484**	30920	**484**	67378
FIFA	25.87	**37.2**	24.44	27.98	**3027**	167935	**3027**	1397238
SIGN	3.54	**34.795**	4.11	10.14	**262**	4477	**262**	180396
KOSARAK	31.44	**34.26**	25.3	21.34	**16646**	234301	**16646**	1146462
BIBLE	6.26	82.06	29.06	**82.48**	**75**	11070	**75**	79456

Scalability. Our second experiment compares the scalability of each approach. The importance of scalability is application specific. But it is an important factor for most prediction tasks since the ability to scale of a prediction

Table 3. Comparison of training time and testing time

Dataset	Training time (s)				Testing time (s)			
	DG	CPT	PPM	AKOM	DG	CPT	PPM	AKOM
BMS	0.076	0.018	**0.01**	0.356	0.004	0.352	**0.001**	0.004
FIFA	3.032	0.153	**0.095**	12.347	0.301	0.146	**0.006**	0.085
SIGN	0.172	**0.008**	0.009	0.455	0.002	0.134	**0.001**	0.002
KOSARAK	9.697	0.741	**0.173**	6.051	0.042	1.533	0.018	**0.011**
BIBLE	0.803	**0.007**	0.244	4.031	0.018	0.029	0.043	**0.002**

model can directly or indirectly limit its accuracy and coverage. For this experiment, we used the FIFA dataset because of its high number of sequences and unique items. The experiment is conducted in steps, where the predictors are trained and tested with a higher number of sequences at each following step. Figure 3 shows the results in terms of accuracy, space and time. All training times in this experiment follow a linear evolution, but CPT and PPM operate at a much lower scale and are very close. PPM and DG have low spatial complexity because of their compact representation compared to CPT which takes more place but still grows linearly.

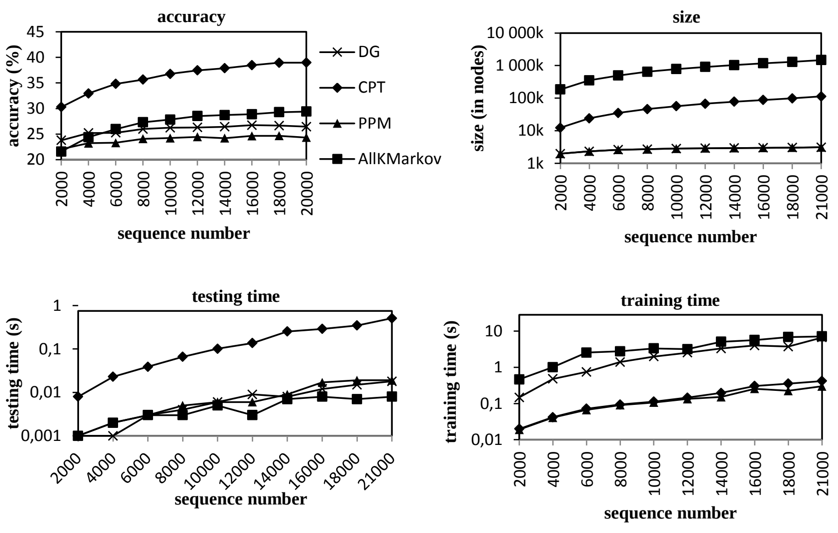


Fig. 3. Comparion of scalability

Prefix length. The third experiment assesses the effect of the prefix size on the accuracy and coverage. Results are shown in Figure 4 for the FIFA dataset. Recall that the predictors output a prediction based on the prefix given as input.

The longer the prefix, the more contextual information is given to the predictor. Note that DG, PPM and All-Kth-Order Markov use a predetermined portion of the prefix defined by the order of each algorithm. Thus, by increasing the prefix length, we can observe that none of these algorithms get an increase in any performance measures. CPT takes advantage of a longer prefix by finding more precise (longer) patterns in its prediction tree, to yield a higher accuracy. The accuracy of CPT gets higher as the prefix length is raised. But after the prefix reaches a length of 8 (specific to the dataset), the accuracy decreases. This is because the algorithm may not be able to match a given prefix to any branches in the prediction tree. It means that this parameter should be finely tuned for each dataset to maximize the accuracy. The figure on the right of Fig. 4 shows the influence of the prefix length on the CPT spatial complexity.

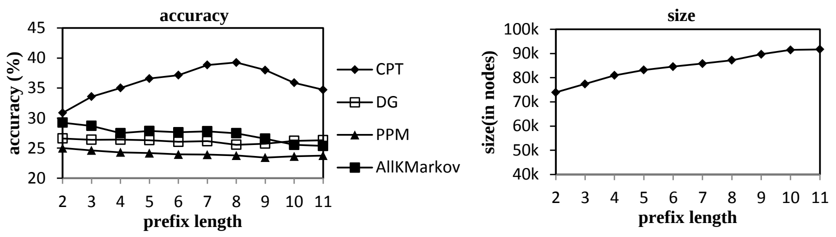


Fig. 4. Influence of prefix length on accuracy and model size

Optimizations. The fourth experiment assesses the influence of the Recursive Divider optimization (cf. Section 3.3) for CPT. The Recursive Divider aims at boosting the coverage of predictions using the CPT by ignoring items that could be noise during the prediction process. But it also indirectly influence accuracy. Figure 5 shows the effect of the Recursive Divider on the FIFA dataset by sequentially incrementing the *maxLevel* parameter. We can observe that the accuracy and the coverage of CPT are getting higher as the *maxLevel* parameter is raised. Also, the coverage and the accuracy measures quickly stabilize without affecting the testing time. This strategy's parameter can therefore be set to a really high value to guarantee the best coverage and accuracy and it does not need to be adjusted for each dataset.

The fifth experiment measures the influence of the Sequence Splitter optimization (cf. Section 3.3). It truncates long sequences before they are inserted in the prediction tree during the training phase. This makes the prediction tree more compact by reducing the number of possible branches and their depth. Reducing the depth improves the time complexity for both the training and testing processes. In this experiment we used the FIFA dataset because it has long sequences. We evaluated the performance of our model against different values for the *splitLength* parameter. For low values (eg. 5) most of the training sequences are split. By setting *splitLength* to a high value (eg. 40 or more), only a small number of sequences are splitted. In Figure 6, we show the effect of applying the

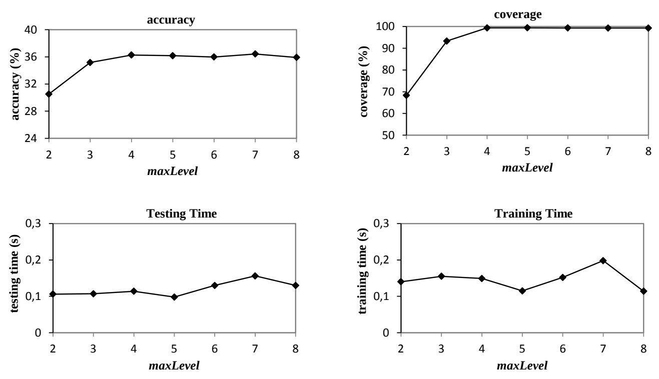


Fig. 5. Influence of the Recursive Divider optimization

Sequence Splitter strategy on the accuracy, the spatial size and the testing time, for various split lengths. By setting *splitLength* to a low value (left side of each chart of Fig. 6), the spatial size is reduced by a factor of 7 while having a really low training and testing time and still having a high accuracy. Once again, this parameter should be finely tuned for each dataset if one wants to achieve the best performances.

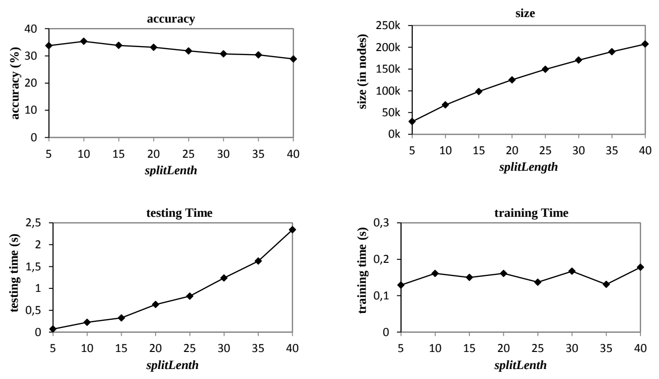


Fig. 6. Influence of the Sequence Splitter optimization

5 Conclusion

Predicting the next item of a sequence over a finite alphabet is essential to a wide range of applications in many domains. In this paper we presented a novel prediction model named the Compact Prediction Tree for sequence prediction. CPT is lossless (it can use all the information from training sequences to make a prediction), is built ly with a low time complexity. We also presented two optimizations (Recursive Divider and Sequence Splitter), which respectively boost the coverage of CPT and reduce its size.

We compared CPT to state-of-the-art approaches, namely PPM, All-Kth-Order Markov Model and DG on six real-life datasets. The source code of algorithms and datasets used in the experiments are available at `http://goo.gl/hDtdt`. Results show that CPT achieves the highest accuracy on all but one dataset with an accuracy up to 12% higher than the second best approach. CPT also shows better training time than DG and All-Kth Order Markov Model by at least a factor of 3. CPT is also considerably smaller than the All-Kth Order Markov model by at least a factor of 2. CPT is easily adaptable for different applications and contexts as shown in the experiments.

In the future, we aim to further improve the accuracy of CPT and its compression. We believe that higher compression can be achieved by grouping patterns of nodes and pruning nodes in the prediction tree. We also plan to compare our model against other prediction techniques such as Context Tree Weighting and Neural Networks.

References

1. Arlitt, M., Jin, T.: A workload characterization study of the 1998 world cup web site. IEEE Network, vol. 14, no. 3, pp. 30-37 (2000)
2. 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)
3. Deshpande, M., Karypis, G.: Selective Markov models for predicting Web page accesses, ACM Transactions on Internet Technology, vol. 4 no. 2, pp. 163-184 (2004)
4. 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)
5. Padmanabhan, V.N., Mogul, J.C.: Using Prefetching to Improve World Wide Web Latency, Computer Communications, vol. 16, pp. 358-368 (1998)
6. Domenech, J., de la Ossa, B., Sahuquillo, J., Gil, J. A., Pont, A.: A taxonomy of web prediction algorithms. In: Expert Systems with Applications, no. 9, (2012)
7. 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)
8. 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)
9. 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)
10. 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)
11. Zheng, Z., Kohavi, R., Mason, L.: Real world performance of association rule algorithms. In: Proc. 7th ACM intern. conf. on KDD, pp. 401-406 (2001)
12. Pitkow, J., Pirolli, P.: Mining longest Repeating subsequences to Predict World Wide Web Surfing. In: 2nd USENIX Symp. Internet Techn. and Systems (1999)