
Good Morning!

Compact Prediction Trees

By Gueniche et al. 2013

Discrete Sequence Prediction: A B C ?

Predict s_{n+1} given a sequence $[s_1, \dots, s_n]$.

or

Find the next element in a sequence based on historical data.

RNNs, 1D CNNs, Markov Chains, Prediction by Partial Matching, ...

Use Cases

Web prefetching - Cache frequently viewed pages.

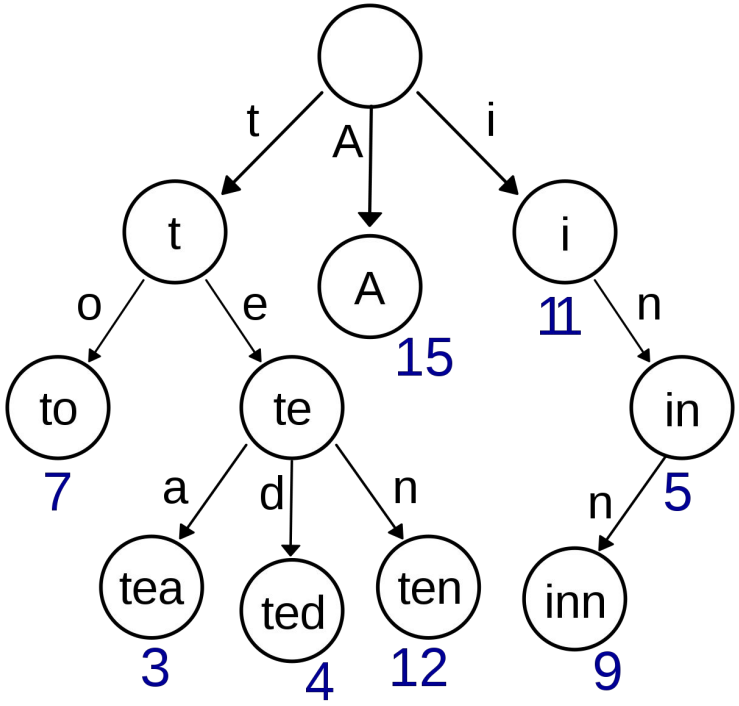
Recommendation engines - What is the customer going to want next?

Forecasting - Unit sales, weather, bitcoin prices, ...?

Why we are here: CPTs

The 3 Building Blocks of Compact Prediction Trees

Data Structures 101 - Trie



https://en.wikipedia.org/wiki/Trie

Storing the Past [PT - Prediction Tree]

Each node is a sequence element

Sequences are stored as Trie branches

Very efficient with common elements

```
struct Node<T> {
  parent: Node,
  children: Vec<Node>,
  value: Option<T>
}
```

Lossless storage of training data

Data Structures 101 - Inverted Index

Values point to documents

Implemented as maps

Fast lookups

Used in search indices (Lucene)

```
["hello world", "hello world",
"hello foo"]
```

```
{
 "hello": [0, 1, 2],
 "world": [0, 1],
 "foo": [2],
}
```

Retrieving the Past [II - Inverted Index]

Each element goes into an inverted index

A “document” is a sequence

Can be implemented as a bitmap

3 Sequences:

```
[ “ABC”, “DEF”, “ABE” ]
```

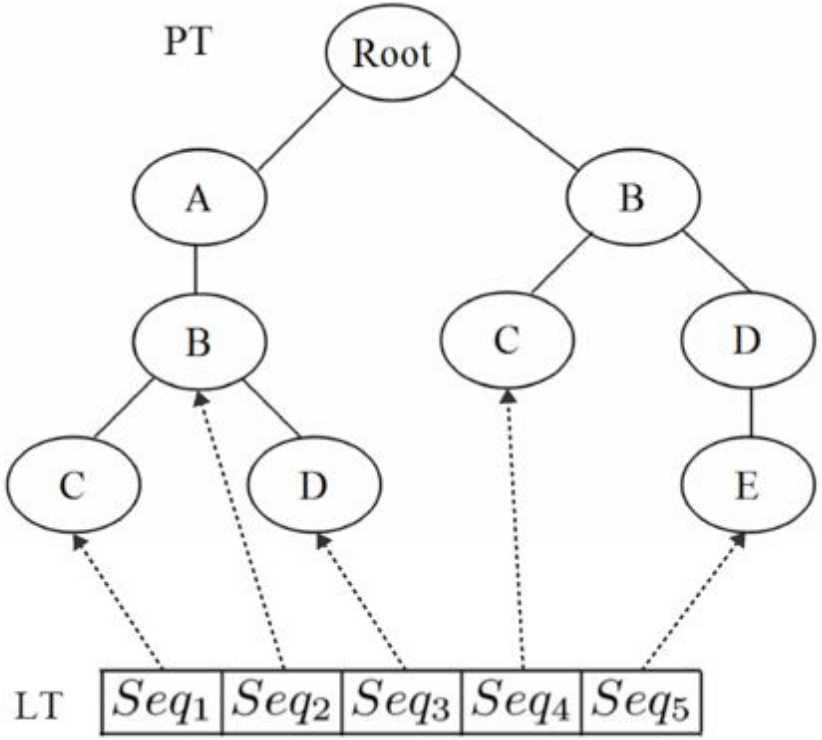
```
{
 “A”: [0, 2],
 “B”: [0, 2],
 “C”: [0],
 “D”: [1],
 “E”: [1],
 “F”: [1]
}
```

Looking Up IDs [LT - Lookup Table]

A simple map to sequences

IDs point to the last sequence element in the trie

Retrieve sequences in reverse



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

From the paper: a trained CPT.

Training And Prediction

Training

1. Insert sequence into the PT
2. Create ID and add to LT
3. Add each element to the II

Time complexity **O(n)** where **n** is the number of training sequences

Prediction

1. Define a lookback number
2. Find all sequences containing **all** lookback elements
3. Get **all** elements that follow after the last lookback element (called *consequent of a training sequence w.r.t to the input sequence*)
4. Create a count table (CT) with these sequences:
    a. Each element is a key with the score as the value
    b. Use confidence and support as scores
5. Pick the highest scoring element

Support and Confidence

Support is the number of appearances in similar consequents

Confidence is support divided by the total number of appearances of in the training sequences (look up in the II).

Optimizations

Sequence Splitter

Introduce a maximum sequence length and split at training

Makes the CPT lossy!

Recursive Divider

Repeatedly remove `1 < maxLevel` elements from the lookback set

Update CT with these elements

Reduces noise influence

Some Numbers

Training

Training time: **O(n)**

PT size **O(n * avgSeqLength)** bytes

II size **O((n + b) * u)** bytes

LT size **O(n * (b + p))** bytes

b = element size, u = unique elements, p = pointer size

Prediction

Prediction time: **O(1) + O(c) + O(m)**

Find common sequences: **O(1)** by using AND in a bitset

CT construction is **O(c)**, picking the best item **O(m)**

c = number of elements in all consequents, m = unique consequent elements

Evaluation

Data and Contestants

DG - Dependency graph

PPM - 1st order Prediction by Part. Matching

AKOM - All-Kth-Order Markov (5th order)

BMS - rule mining data set for the KDD CUP 2000, web sessions for e-commerce

FIFA - Random sample of FIFA web site sessions during the 1998 FIFA World Cup

SIGN - Sign language transcriptions from videos

KOSARAK - Web sessions from a Hungarian news site (http://fimi.ua.ac.be/data)

BIBLE - The holy bible's characters in sentences

Framework

Various element data types

Split into 3 parts: context, prefix, suffix

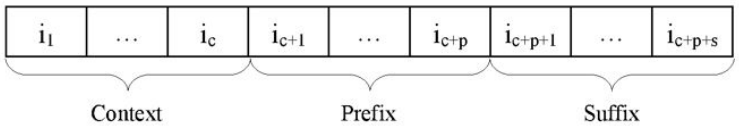


Fig. 2. Sequence splitting (context, prefix, suffix)

Success is when a prediction appears in the suffix

Failure is when the it did not

No_match is when no prediction was made

Accuracy = |*success*| / |*sequences*|

Local Accuracy = |*success*| / (|*success*| + |*failure*|)

Coverage = |*no_match*| / |*sequences*|

Table 1. Dataset characteristics

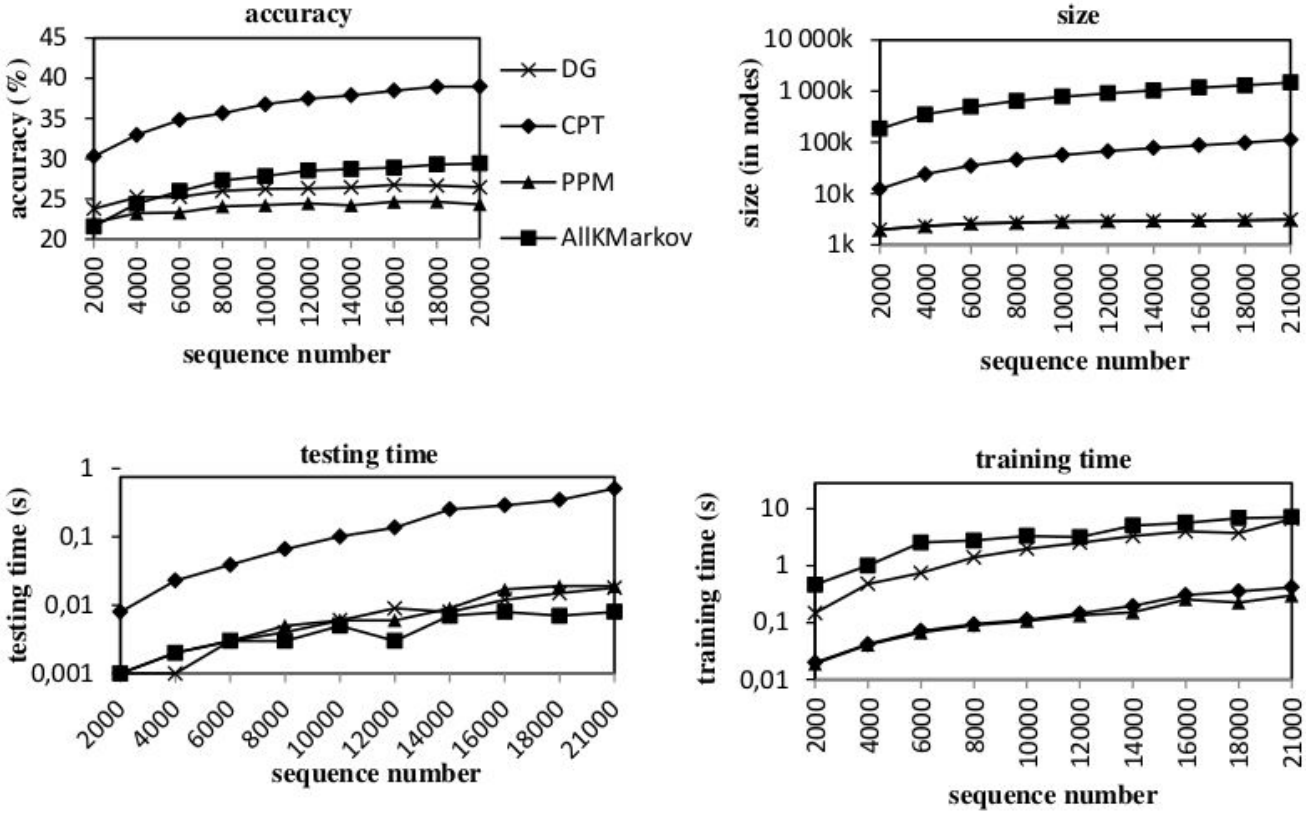
Dataset	**Sequence count**	**Unique items**	**Avg sequence length**	**Avg item occurence count per sequence**
BMS	15,806	495	6.01	1.00
FIFA	28,978	3,301	32.11	1.04
SIGN	730	267	93.00	1.79
KOSARAK	638,811	39,998	11.64	1.00
BIBLE	32,529	76	130.96	4.78

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

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**



FIFA

Fig. 3. Comparion of scalability

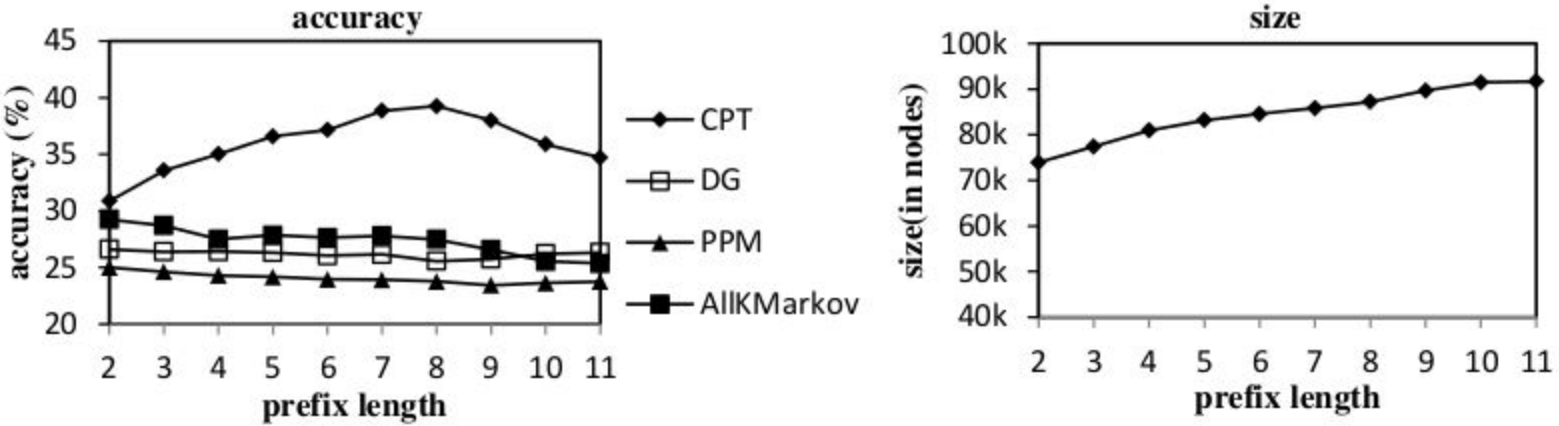


Fig. 4. Influence of prefix length on accuracy and model size

FIFA

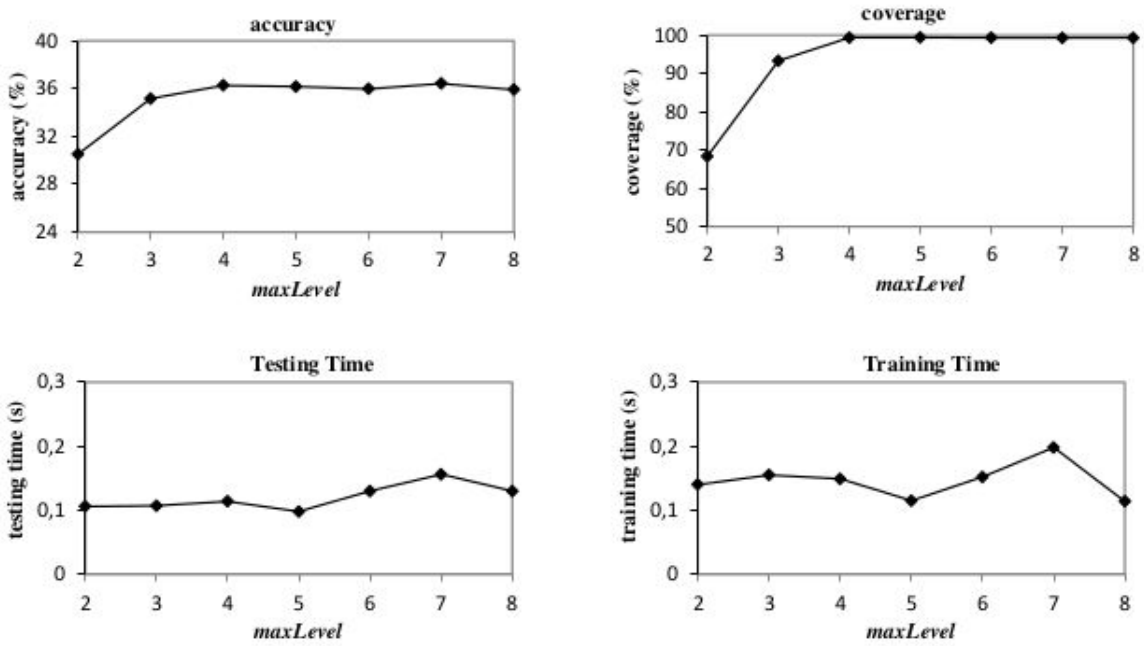


Fig. 5. Influence of the Recursive Divider optimization

FIFA

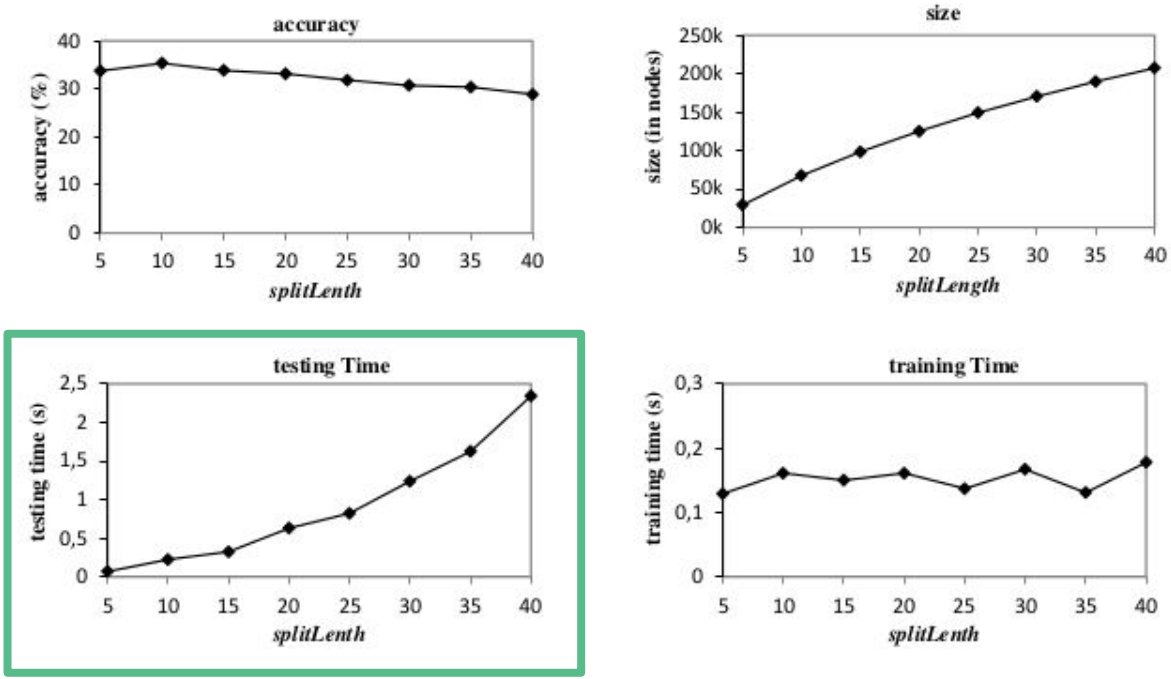


Fig. 6. Influence of the Sequence Splitter optimization

FIFA

BIBLE

The holy bible!

Corpus of religious Christian books

75 distinct characters, natural language

Found on Kaggle

```
$ node index.mjs
Training on 27991 sequences, testing on 3111.
Training time: 2s 91.811089ms
Prediction time: 581s 240.295153ms
(0.18675666988106718s per item)
Local Accuracy: 0.6665594855305467, coverage:
0.0003214400514304082, accuracy:
0.6663452266152362
Sample:  [ 'a', 'l', 'l', ',', ' ', 's', 'a',
'i', 't', 'h' ] , actual  [ ' ', 't', 'h',
'e', ' ', 'L', 'O', 'R', 'D', '.' ]  vs
predicted  [ 'e' ]
```

Overall Performance and Outcome

Higher accuracy, better consistency

Great training times, but long testing/prediction

Small model size, flexible usage

CPT is a lossless predictor for finite alphabets

CPT+ is out!

Resources

blog.x5ff.xyz - Watch for a Rust implementation :)

CPT+ - Improved version of the CPT by the same authors

github.com/ashubham/CPT - A JavaScript implementation

https://www.analyticsvidhya.com/blog/2018/04/guide-sequence-prediction-using-compact-prediction-tree-python/ - A Python implementation

http://www.philippe-fournier-viger.com/spmf/ - Author’s website and Java implementation

http://aka.ms/talking-ai-podcast - My CSE AI podcast

Discussion!