CPT+: A Compact Model for Accurate Sequence Prediction

Ted Gueniche¹, Philippe Fournier-Viger¹, Rajeev Raman², Vincent S. Tseng³

¹University of Moncton, Canada
²University of Leicester, UK
³National Chiao Tung University, Taiwan
The problem of Sequence Prediction

- **Problem:**
  - Given a set of training sequences, predict the next symbol of a sequence.

- **Applications:**
  - webpage prefetching,
  - analyzing the behavior of customers on websites,
  - keyboard typing prediction
  - product recommendation,
  - stock market prediction,
  - …
General approach for this problem

Phase 1) Training

- Training sequences
- Building a sequence prediction model
- Prediction Model

Phase 2) Prediction

- Prediction Model
- Prediction algorithm
- A sequence e.g. A,B,C
- Prediction e.g. D
Sequential pattern mining

- Discovery of patterns
- Using the patterns for prediction
- It is time-consuming to extract patterns.
- patterns ignore rare cases,
- updating the patterns: very costly!

<table>
<thead>
<tr>
<th>ID</th>
<th>sequences</th>
<th>Pattern</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>a, b, {c d}, e</td>
<td>a</td>
<td>83 %</td>
</tr>
<tr>
<td>S2</td>
<td>a, d, e</td>
<td>a, e</td>
<td>66 %</td>
</tr>
<tr>
<td>S3</td>
<td>e, f, g</td>
<td>b, c</td>
<td>33 %</td>
</tr>
<tr>
<td>S4</td>
<td>b, a, d, e</td>
<td>a, d, e</td>
<td>50 %</td>
</tr>
<tr>
<td>S5</td>
<td>{a, b}, {c d}, f, g</td>
<td>a, {c, d}</td>
<td>33 %</td>
</tr>
<tr>
<td>S6</td>
<td>a, e</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

PrefixSpan

Minsup = 33 %
Dependency Graph (DG)

S_1: \{A, B, C, A, C, B, D\}

DG with lookup table of size 2
Dependency Graph (DG)

$S_1$: \{A, B, C, A, C, B, D\}


$P(B \mid A) = \frac{3}{\text{SUP}(A)} = \frac{3}{4}$

$P(C \mid A) = \frac{3}{\text{SUP}(A)} = \frac{3}{4}$

...
PPM – order 1
(prediction by partial matching)

$S_1$: \{A,B,C,A,C,B,D\}
PPM – order 1
(prediction by partial matching)

$S_1$: \{A,B,C,A,C,B,D\}

\[
P(B | A) = \frac{2}{4}
\]
\[
P(C | A) = \frac{1}{4}
\]

\[
P(B | A) = \frac{2}{4}
\]
\[
P(C | A) = \frac{1}{4}
\]

\[
\ldots
\]
PPM – order 2

$S_1$: \{A,B,C,A,C,B,D\}

predictions are inaccurate if there is noise…
All-K-Order Markov

- Uses PPM from level 1 to K for prediction.
- More accurate than a fixed-order PPM,
- But exponential size

Example: order 2

- $P(C \mid AB) = \frac{2}{2}$
- $P(B \mid AC) = \frac{1}{1}$
- $P(A \mid BC) = \frac{2}{3}$

...
Limitations

- Several models assume that each event depends only on the immediately preceding event.
- Otherwise, often an exponential complexity (e.g.: All-K-Markov)
- Some improvements to reduce the size of markovian models, but few work to improve their accuracy.
- Several models are not noise tolerant.
- Some models are costly to update (e.g. sequential patterns).
- All the aforementioned models are lossy models.
Goal

◦ to provide more accurate predictions,
◦ a model having a reasonable size,
◦ a model that is noise tolerant.
Hypothesis

• **Idea:**
  ◦ build a lossless model (*or* a model where the loss of information can be controlled),
  ◦ use all relevant information to perform each sequence prediction.

• **Hypothesis:**
  ◦ this would increase prediction accuracy.
Challenges

1) Define an efficient structure in terms of space to store sequences,

2) The structure must be incrementally updatable to add new sequences

3) Propose a prediction algorithm that:
   ◦ offers accurate predictions,
   ◦ if possible, is also time-efficient.
Our proposal

Compact Prediction Tree (CPT)

- A tree-structure to store training sequences,
- An indexing mechanism,
- Each sequence is inserted one after the other in the CPT.

Illustration ➔
Example

We will consider the four following training sequences:

1. ABC
2. AB
3. ABDC
4. BC
5. BDE
Example (construction)

Prediction tree
root

Inverted Index

Lookup table
Example: Inserting \(<A,B,C>\)

Prediction tree

root

Inverted Index

Lookup table
Example: Inserting \(<A,B,C>\)

**Prediction tree**

- **root**
  - A
    - B
      - C

**Inverted Index**

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

**Lookup table**

<table>
<thead>
<tr>
<th>s1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Example: Inserting <A,B>

Prediction tree

root

A

B

C

Lookup table

<table>
<thead>
<tr>
<th>s1</th>
<th>s2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

Inverted Index
Example: Inserting \(<A,B,D,C>\)

![Prediction tree diagram]

Inverted Index

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Inserting $<B, C>$

Prediction tree:
- Root
- A
  - B
    - C

Inverted Index:

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: Inserting <B,D,E>

Prediction tree

Inverted Index

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Inserting <B,D,E>

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Insertion

- **linear complexity**, $O(m)$ where $m$ is the sequence length.
- a reversible operation (sequences can be recovered from the CPT).
- **the insertion order** of sequences is preserved in the CPT.
Space complexity

Size of the prediction tree

- worst case: $O(N \times \text{average sequence length})$ where $N$ is the number of sequences.
- In general, much smaller, because sequences overlap.
Space complexity (cont’d)

Size of Inverted Index

- \((n \times b)\)
  - \(n\) = sequence count
  - \(b\) = symbol count

- small because encoded as bit vectors

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Space complexity (cont’d)

Size of lookup table

\( n \) pointers where \( n \) is the sequence count

Lookup table

\begin{array}{ccccc}
\text{s1} & \text{s2} & \text{s3} & \text{s4} & \text{s5} \\
\end{array}
PREDICTION
Predicting the symbol following \(<A,B>\)
Predicting the symbol following \(<A,B>\)

**Prediction tree**

- **Root**
  - **A**
    - **B**
      - **C**
      - **D**
      - **E**
    - **D**
  - **C**
  - **D**
  - **E**

**Inverted Index**

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The logical AND indicates that the sequences common to A and B are: s1, s2 et s3
Predicting the symbol following \textless A,B\textgreater

The \textit{Lookup table} allows to traverse the corresponding sequences in from the end to the start.
Predicting the symbol following \(<A, B>\)

**Prediction tree**

- **Root**
- **A**
  - **B**
    - **C**
    - **D**
  - **E**

**Inverted Index**

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Count table:**
- **C**: 2 occurrences after \{AB\}
- **D**: 1 occurrences after \{AB\}
Complexity of prediction

1. **Intersection of bit vectors**: \( O(v) \) where \( v \) is the number of symbols.

2. **Traversing sequences**: \( O(n) \) where \( n \) is the sequence count.

3. **Creating the count table**: \( O(x) \) where \( x \) is the number of symbols in sequences after the target sequence.

4. **Choosing the predicted symbol**: \( O(y) \) where \( y \) is the number of distinct symbols in the Count Table.
EXPERIMENTAL EVALUATION
Experimental evaluation

Datasets

- **BMS, FIFA, Kosarak**: sequences of clicks on webpages.
- **SIGN**: sentences in sign languages.
- **BIBLE**: sequences of characters in a book.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Sequence count</th>
<th>Unique items</th>
<th>Avg sequence length</th>
<th>Avg item occurrence count per sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS</td>
<td>15,806</td>
<td>495</td>
<td>6.01</td>
<td>1.00</td>
</tr>
<tr>
<td>FIFA</td>
<td>28,978</td>
<td>3,301</td>
<td>32.11</td>
<td>1.04</td>
</tr>
<tr>
<td>SIGN</td>
<td>730</td>
<td>267</td>
<td>93.00</td>
<td>1.79</td>
</tr>
<tr>
<td>KOSARAK</td>
<td>638,811</td>
<td>39,998</td>
<td>11.64</td>
<td>1.00</td>
</tr>
<tr>
<td>BIBLE</td>
<td>32,529</td>
<td>76</td>
<td>130.96</td>
<td>4.78</td>
</tr>
</tbody>
</table>
Experimental evaluation (cont’d)

Competitor algorithms

- DG (lookup window = 4)
- All-K-Order Markov (order of 5)
- PPM (order of 1)

10-fold cross-validation
Experimental evaluation (cont’d)

Measures:

- **Accuracy**
  \[ \frac{|\text{success count}|}{|\text{sequence count}|} \]
- **Coverage**
  \[ \frac{|\text{prediction count}|}{|\text{sequence count}|} \]
**Experiment 1 – Accuracy**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>DG</th>
<th>CPT</th>
<th>PPM</th>
<th>AKOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS</td>
<td>36.07</td>
<td>38.45</td>
<td>31.12</td>
<td>30.81</td>
</tr>
<tr>
<td>FIFA</td>
<td>25.87</td>
<td>37.2</td>
<td>24.44</td>
<td>27.98</td>
</tr>
<tr>
<td>SIGN</td>
<td>3.54</td>
<td>34.795</td>
<td>4.11</td>
<td>10.14</td>
</tr>
<tr>
<td>KOSARAK</td>
<td>31.44</td>
<td>34.26</td>
<td>25.3</td>
<td>21.34</td>
</tr>
<tr>
<td>BIBLE</td>
<td>6.26</td>
<td>82.06</td>
<td>29.06</td>
<td>82.48</td>
</tr>
</tbody>
</table>

- **CPT** is the most accurate except for one dataset.
- **PPM** and **DG** perform well in some situations.
Experiment 1 – size

<table>
<thead>
<tr>
<th>Dataset</th>
<th>DG</th>
<th>CPT</th>
<th>PPM</th>
<th>AKOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS</td>
<td>484</td>
<td>30920</td>
<td>484</td>
<td>67378</td>
</tr>
<tr>
<td>FIFA</td>
<td>3027</td>
<td>167935</td>
<td>3027</td>
<td>1397238</td>
</tr>
<tr>
<td>SIGN</td>
<td>262</td>
<td>4477</td>
<td>262</td>
<td>180396</td>
</tr>
<tr>
<td>KOSARAI</td>
<td>16646</td>
<td>234301</td>
<td>16646</td>
<td>1146462</td>
</tr>
<tr>
<td>BIBLE</td>
<td>75</td>
<td>11070</td>
<td>75</td>
<td>79456</td>
</tr>
</tbody>
</table>

- **CPT** is
  - smaller than All-K-order-Markov
  - larger than DG and PPM
- CPT's training time is at least 3 times less than DG and AKOM, and similar to PPM.
- CPT's prediction time is quite high (a trade-off for more accuracy)
CPT shows a trend similar to other algorithms.
Experiment 3 – prefix size

- **prefix size**: the number of symbols to be used for making a prediction
- for FIFA:

The accuracy of CPT increases until a prefix size of around 8. (depends on the dataset)
Optimisation #1 - RecursiveDivider

Example: \{A,B,C,D\}

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>{B,C,D}</td>
<td>{C,D}</td>
<td>{D}</td>
</tr>
<tr>
<td>{A,C,D}</td>
<td>{B,D}</td>
<td>{C}</td>
</tr>
<tr>
<td>{A,B,D}</td>
<td>{B,C}</td>
<td>{B}</td>
</tr>
<tr>
<td>{A,B,C}</td>
<td>{A,D}</td>
<td>{A}</td>
</tr>
<tr>
<td>{A,C}</td>
<td>{A,B}</td>
<td></td>
</tr>
</tbody>
</table>

Accuracy and coverage are increasing.

Training time and prediction time remains more or less the same.

Therefore, a high value for this parameter is better for all datasets.
Optimisation #2 – sequence splitting

Example:
splitting sequence \{A,B,C,D,E,F,G\} with \textit{split\_length} = 5 gives \{C,D,E,F,G\}
Conclusion

- **CPT, a new model for sequence prediction**
  - allows fast incremental updates,
  - compresses training sequences,
  - integrates an indexing mechanism
  - two optimizations,

- **Results:**
  - in general, more accurate than compared models but prediction time is greater (a trade-off),
  - CPT is more than twice smaller than AKOM
  - sequence insertion more than 3 times faster than DG and AKOM
CPT+: DECREASING THE TIME/SPACE COMPLEXITY OF CPT

Introduction

- Two optimisations to reduce the size of the tree used by CPT:
  - compressing frequent substrings,
  - compressing simple branches.
- An optimisation to improve prediction time and noise tolerance.
(1) compressing frequent substrings

- This strategy is applied during training
  - it identifies frequent substrings in training sequences,
  - it replaces these substrings by new symbols
- Discovering substrings is done with a modified version of the PrefixSpan algorithm
  - parameters: \textit{minsup}, \textit{minLength} and \textit{maxLength}
(1) compressing frequent substrings

Prediction tree

Inverted Index

Lookup table
(1) compressing frequent substrings

Prediction tree

Inverted Index

Lookup table

\[ x = AB \]
(1) Compressing simple branches

- **Time complexity:**
  - training: non negligible cost to discover frequent substrings,
  - prediction: symbols are uncompressed on-the-fly in $O(1)$ time.

- **Space complexity:**
  - $O(m)$ where $m$ is the number of frequent substrings.
(2) Compressing simple branches

- A second optimization to reduce the size of the tree
- A *simple branch* is a branch where all nodes have a single child.
- Each simple branch is replaced by a single node representing the whole branch.
(2) Compressing simple branches

Prediction tree

Inverted Index

\[
x = AB
\]

Lookup table
(2) Compressing simple branches

Prediction tree

Inverted Index

<table>
<thead>
<tr>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ x = AB \]

Lookup table
(2) Compressing simple branches

Prediction tree

Inverted Index

Lookup table

x = AB
(2) Compressing simple branches

- Time complexity
  - very fast.
  - after building the tree, we only need to traverse the branches from the bottom using the lookup table.
(3) Improved Noise Reduction

• Recall that CPT removes items from a sequence to be predicted to be more noise tolerant.

• Improvement:
  ◦ only remove less frequent symbols from sequences, assuming that they are more likely to be noise,
  ◦ consider a minimum number of sequences to perform a prediction,
  ◦ add a new parameter **Noise Ratio** (e.g. 20%) to determine how many symbols should be removed from sequences (e.g.: the 20% most infrequent symbols).
  ◦ Thus, the amount of noise is assumed to be proportional to the length of sequences.
### Experiment

#### Datasets

<table>
<thead>
<tr>
<th>Name</th>
<th>Sequence count</th>
<th>Distinct item count</th>
<th>Average length</th>
<th>Type of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS</td>
<td>15,806</td>
<td>495</td>
<td>6.01</td>
<td>webpages</td>
</tr>
<tr>
<td>KOSARAK</td>
<td>638,811</td>
<td>39,998</td>
<td>11.64</td>
<td>webpages</td>
</tr>
<tr>
<td>FIFA</td>
<td>573,060</td>
<td>13,749</td>
<td>45.32</td>
<td>webpages</td>
</tr>
<tr>
<td>MSNBC</td>
<td>250,697</td>
<td>17</td>
<td>3.28</td>
<td>webpages</td>
</tr>
<tr>
<td>SIGN</td>
<td>730</td>
<td>267</td>
<td>93.00</td>
<td>language</td>
</tr>
<tr>
<td>BIBLE Word</td>
<td>42,436</td>
<td>76</td>
<td>18.93</td>
<td>sentences</td>
</tr>
<tr>
<td>BIBLE Char</td>
<td>32,502</td>
<td>75</td>
<td>128.35</td>
<td>characters</td>
</tr>
</tbody>
</table>

#### Competitor algorithms

DG, TDAG, PPM, LZ78, All-K-Markov
## Prediction accuracy

<table>
<thead>
<tr>
<th>dataset</th>
<th>CPT+</th>
<th>CPT</th>
<th>AKOM</th>
<th>DG</th>
<th>LZ78</th>
<th>PPM</th>
<th>TDAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMS</td>
<td>38.25</td>
<td>37.90</td>
<td>31.26</td>
<td>36.46</td>
<td>33.46</td>
<td>31.06</td>
<td>6.95</td>
</tr>
<tr>
<td>SIGN</td>
<td>33.01</td>
<td>32.33</td>
<td>8.63</td>
<td>3.01</td>
<td>4.79</td>
<td>4.25</td>
<td>0.00</td>
</tr>
<tr>
<td>MSNBC</td>
<td>61.50</td>
<td>61.64</td>
<td>47.88</td>
<td>55.68</td>
<td>43.64</td>
<td>38.06</td>
<td>31.14</td>
</tr>
<tr>
<td>Bible word</td>
<td>27.52</td>
<td>22.05</td>
<td>38.68</td>
<td>24.92</td>
<td>27.39</td>
<td>27.06</td>
<td>11.17</td>
</tr>
<tr>
<td>Bible char</td>
<td>73.52</td>
<td>69.14</td>
<td>7.96</td>
<td>0.00</td>
<td>3.02</td>
<td>0.10</td>
<td>0.00</td>
</tr>
<tr>
<td>Kosarak</td>
<td>37.64</td>
<td>33.82</td>
<td>20.52</td>
<td>30.82</td>
<td>20.50</td>
<td>23.86</td>
<td>1.06</td>
</tr>
<tr>
<td>FIFA</td>
<td>35.94</td>
<td>34.56</td>
<td>25.88</td>
<td>24.78</td>
<td>24.64</td>
<td>22.84</td>
<td>7.14</td>
</tr>
</tbody>
</table>

**CPT+ is also up to 4.5 times faster than CPT in terms of prediction time**
Scalability

The diagram shows the relationship between sequence count and size (nodes) for various algorithms: DG, TDAG, PPM, LZ78, CPT, and CPT+. The x-axis represents the sequence count, and the y-axis represents the size in nodes. The graph illustrates how each algorithm scales with increasing sequence count, with DG and TDAG showing more significant increases compared to PPM, LZ78, CPT, and CPT+.
Conclusion

- CPT(+): a novel sequence prediction model
- Fast training time
- Good scalability,
- High prediction accuracy.

Future work:
- further compress the model,
- compare with other predictions models such as CTW and NN,
- data stream, user profiles…
- open-source library for web prefetching
  IPredict