Suffix Array and Its Applications in Empirical Natural Language Processing
Tutorial of the SALM package

Joy Ying Zhang
joy+@cs.cmu.edu

Language Technologies Institute
School of Computer Science
Carnegie Mellon University

Jan. 16, 2007
n-gram Statistics in Empirical NLP

- n-gram based language model: \( P(e_i|e_{i-n+1}^{i-1}) \)
- Phrase (n-gram) based Statistical Machine Translation (SMT)
  - Sample training corpus for testing data
  - Find occurrences of n-gram in training data
- Word frequency list used in Chinese word segmentation: a word as a character n-gram
- Term frequency / document frequency in \( tf \cdot idf \) for Information Retrieval (IR)
Straightforward Implementations Won’t Work

- Big deal? 5 lines of perl scripts with a hash list and done!
- Yes, 5-line script can do all the jobs for a corpus of a few million words.
- We are talking about billions of words.

<table>
<thead>
<tr>
<th>n</th>
<th>Type</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1,607,516</td>
<td>553,290,787</td>
</tr>
<tr>
<td>2</td>
<td>23,449,845</td>
<td>538,622,854</td>
</tr>
<tr>
<td>3</td>
<td>105,752,368</td>
<td>523,954,921</td>
</tr>
<tr>
<td>4</td>
<td>221,359,119</td>
<td>509,287,302</td>
</tr>
<tr>
<td>5</td>
<td>323,180,272</td>
<td>509,288,452</td>
</tr>
<tr>
<td>6</td>
<td>383,721,940</td>
<td>509,291,400</td>
</tr>
<tr>
<td>7</td>
<td>416,639,942</td>
<td>509,297,474</td>
</tr>
<tr>
<td>8</td>
<td>435,223,983</td>
<td>509,313,484</td>
</tr>
<tr>
<td>9</td>
<td>446,820,635</td>
<td>509,340,447</td>
</tr>
<tr>
<td>10</td>
<td>454,850,677</td>
<td>509,386,608</td>
</tr>
</tbody>
</table>

Number of $n$-gram types and tokens in a corpus of 600M words.
Why USSR Failed to Send a Man to the Moon?

- USSR was the first to send a man into space.
Why USSR Failed to Send a Man to the Moon?

- USSR was the first to send a man into space.
- USSR had the “scientific” ideas in the space race.
Why USSR Failed to Send a Man to the Moon?

- USSR was the first to send a man into space.
- USSR had the “scientific” ideas in the space race.
- But USSR lost the space race to U.S.A
Why USSR Failed to Send a Man to the Moon?

- USSR was the first to send a man into space.
- USSR had the “scientific” ideas in the space race.
- But USSR lost the space race to U.S.A
- Why? USSR could not develop a powerful rocket to send the spacecraft to the lunar orbit.
Why USSR Failed to Send a Man to the Moon?

▶ USSR was the first to send a man into space.
▶ USSR had the “scientific” ideas in the space race.
▶ But USSR lost the space race to U.S.A
▶ Why? USSR could not develop a powerful rocket to send the spacecraft to the lunar orbit.
▶ NASA successfully developed Saturn-V rockets, a reliable powerful platform to carry out the ambitious plans
Why USSR Failed to Send a Man to the Moon?

- USSR was the first to send a man into space.
- USSR had the “scientific” ideas in the space race.
- But USSR lost the space race to U.S.A
- Why? USSR could not develop a powerful rocket to send the spacecraft to the lunar orbit.
- NASA successfully developed Saturn-V rockets, a reliable powerful platform to carry out the ambitious plans

Good engineering makes great ideas come true!
SALM: Suffix Array Library and Toolkit for EMNLP

- Index large corpus (up to 4 billion words) based on suffix array
- Efficient search for arbitrarily long $n$-grams without storing all the $n$-grams separately
- Many useful applications developed based on SALM
  - Filter out duplicate sentences in a corpus
  - Assigning frequencies to a Chinese word list
  - Subsample phrase table for a testing data
  - $n$-gram coverage analysis of the testing data given the training corpus
  - Suffix array language model
  - ...
- A great platform for EMNLP research.
Outline

- Suffix array indexing
- Search suffix array
- Scan suffix array
- Suffix array language model
## Suffices in a Corpus

### Corpus \( A \): \( a_1, a_2, \ldots, a_N \)

<table>
<thead>
<tr>
<th>Word Position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word ( a_{pos} )</td>
<td>how</td>
<td>do</td>
<td>you</td>
<td>say</td>
<td>how</td>
<td>do</td>
<td>you</td>
<td>do</td>
<td>in</td>
<td>chinese</td>
</tr>
</tbody>
</table>

### Suffixes:

- \( A_1 \): how do you say how do you do in chinese
- \( A_2 \): do you say how do you do in chinese
- \( A_3 \): you say how do you do in chinese
- \( A_4 \): say how do you do in chinese
- \( A_5 \): how do you do in chinese
- \( A_6 \): do you do in chinese
- \( A_7 \): you do in chinese
- \( A_8 \): do in chinese
- \( A_9 \): in chinese
- \( A_{10} \): chinese
Suffix Array: Sorted Suffices

|  $A_{10}$:  | Chinese                        |
|  $A_{8}$:    | Do in Chinese                  |
|  $A_{6}$:    | Do you do in Chinese          |
|  $A_{2}$:    | Do you say how do you do in Chinese |
|  $A_{5}$:    | How do you do in Chinese      |
|  $A_{1}$:    | How do you say how do you do in Chinese |
|  $A_{9}$:    | In Chinese                    |
|  $A_{4}$:    | Say how do you do in Chinese  |
|  $A_{7}$:    | You do in Chinese             |
|  $A_{3}$:    | You say how do you do in Chinese |

**Suffix Array $X$:**

<table>
<thead>
<tr>
<th>Index: $k$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X[k]$</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>
First step of all applications: index the corpus

Synopsis:

```
IndexSA.O32 corpusFileName [existingIDVocFileName]
```

Inserts ⟨s⟩ and ⟨/s⟩ into each sentence.

Automatically create corpus/suffix array/offset/vocabulary.

Sentence in corpus can not be longer than 256 words.

Corpus size $N$: up to 4 billion words for a single corpus.

RAM needed: $9N$ bytes
Frequency of an $n$-gram

- Binary search in the sorted suffix array
- Search for a range $[L, R]$ in the suffix array such that corresponding suffices all have the $n$ prefix words as the $n$-gram
- Frequency $= R - L + 1$.
- Complexity $O(n \log N)$
Example

\[ A_{10}: \text{chinese} \]
\[ A_8: \text{do in chinese} \]
\[ A_6: \text{do you do in chinese} \]
\[ A_2: \text{do you say how do you do in chinese} \]
\[ A_5: \text{how do you do in chinese} \]
\[ A_1: \text{how do you say how do you do in chinese} \]
\[ A_9: \text{in chinese} \]
\[ A_4: \text{say how do you do in chinese} \]
\[ A_7: \text{you do in chinese} \]
\[ A_3: \text{you say how do you do in chinese} \]

Suffix Array \( X \):

\[
\begin{array}{cccccccccccc}
\text{Index: } k & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
X[k] & 10 & 8 & 6 & 2 & 5 & 1 & 9 & 4 & 7 & 3 \\
\end{array}
\]

▶ *n-gram:* do you

▶ Search result range \([L, R] = [3, 4]\)
Frequencies of Embedded $n$-grams in a Sentence

- A testing sentence of length $J$ has $J(J+1)/2$ embedded $n$-grams
- Search frequencies for all the embedded $n$-gram requires $O(J^3 \log N)$
- Smarter way:
  - The range of *do you* [3, 4] has to be subset of range for *do* [2, 4]
  - Reuse the search results from the shorter $n$-grams when search for long $n$-grams
  - Complexity: $O(J^2 \log N)$

method: displayNgramMatchingFreq4Sent()

method: constructNgramSearchTable4SentWithLCP()
Convert an $n$-gram Position to SentenceID+Offset

- From suffix array search we know positions of an $n$-gram’s occurrences in the corpus
- Convert the position into sentenceID+offset in sentence might be useful
- Insert the sentenceID before $\langle s \rangle$ in each sentence
- Naive method: move from matched position toward sentence start one word at a time
## Example

<table>
<thead>
<tr>
<th>Corpus $\mathcal{F}$: $a_1, a_2, \ldots, a_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
</tr>
<tr>
<td>$a_{pos}$</td>
</tr>
<tr>
<td>Pos</td>
</tr>
<tr>
<td>$a_{pos}$</td>
</tr>
<tr>
<td>Pos</td>
</tr>
<tr>
<td>$a_{pos}$</td>
</tr>
</tbody>
</table>

**Figure:** Augment the corpus with sentence ID before indexing
Convert an $n$-gram Position to SentenceID+Offset

- From suffix array search we know positions of an $n$-gram’s occurrences in the corpus
- Convert the position into sentenceID+offset in sentence might be useful
- Insert the sentenceID before ⟨s⟩ in each sentence
- Naive method: move from matched position toward sentence start one word at a time
- Smarter way:
  - Introduce additional data structure “offset” array
  - From matched position jump directly before ⟨s⟩ to read sentenceID
Corpus $\mathcal{F}$: $a_1, a_2, \ldots, a_N$

<table>
<thead>
<tr>
<th>Pos</th>
<th>$a_{pos}$</th>
<th>$\delta[\text{pos}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 ⟨s⟩</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>these words</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>are supposed to be foreign words ⟨/s⟩</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pos</th>
<th>$a_{pos}$</th>
<th>$\delta[\text{pos}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>2 ⟨s⟩</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>in order to make it easy to read ⟨/s⟩</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>18</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>19</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>20</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>21</td>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pos</th>
<th>$a_{pos}$</th>
<th>$\delta[\text{pos}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>3 ⟨s⟩</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>we use english words here ⟨/s⟩</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>26</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>27</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>28</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>29</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>9</td>
<td>7</td>
</tr>
</tbody>
</table>

**Figure:** Using $\delta$ array for offset information

- sentenceID for $pos$ in corpus is: $a_{pos} - \delta[\text{pos}]$
- offset of $pos$ in the sentence is: $\delta[\text{pos}]$
- Sentences in corpus are shorter than 256 words
- $N$ additional bytes needed for the offset information
Focus on Limited Occurrences of Interested n-grams

- Way too many occurrences of “the” in the corpus
- No need and too expensive to return all its locations in the corpus
- Return limited occurrences:
  - setParam_reportMaxOccurrenceOfOneNgram()
- Focus on interested $n$-grams in the sentence:
  - setParam_highestFreqThresholdForReport()
  - setParam_shortestUnitToReport()
  - setParam_longestUnitToReport()
Filter Out Duplicated Sentences in a Corpus

Program: FilterDuplicatedSentences

- A corpus may contain duplicated sentences
- Index the corpus with its suffix array
- For each sentence, search for its complete match
- If occurs more than once,
  - Check if it has been output already
  - If not, output it and save in memory sentence: posInCorpus has been output.
Scan Through the Suffix Array

- To obtain information regarding the $n$-gram types
- Use a window of size $n$ to scan through the sorted suffix list
- When the $n$-gram type changes during the scanning, we observe a new type
- Linear time to scan through the whole corpus
Suffix Array: Sorted Suffices

\[ A_{10}: \text{chinese} \]
\[ A_8: \text{do in chinese} \]
\[ A_6: \text{do you do in chinese} \]
\[ A_2: \text{do you say how do you do in chinese} \]
\[ A_5: \text{how do you do in chinese} \]
\[ A_1: \text{how do you say how do you do in chinese} \]
\[ A_9: \text{in chinese} \]
\[ A_4: \text{say how do you do in chinese} \]
\[ A_7: \text{you do in chinese} \]
\[ A_3: \text{you say how do you do in chinese} \]

<table>
<thead>
<tr>
<th>chinese</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>do</td>
<td>in</td>
<td>chinese</td>
</tr>
<tr>
<td>do</td>
<td>you</td>
<td>do</td>
</tr>
<tr>
<td>do</td>
<td>you</td>
<td>say</td>
</tr>
<tr>
<td>how</td>
<td>do</td>
<td>you</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Applications based on Suffix Array Scanning

When the $n$-gram type in the scanning window changes, we know the frequency of this $n$-gram type:

- Output the type if we are interested in high frequency $n$-gram types. e.g. Program: OutputHighFreqNgram
- Accumulate count-of-count for $n$-gram. e.g. Program: CalcCountOfCounts
- Accumulate type/token ratio for $n$-gram. e.g. Program: TypeTokenFreqInCorpus
Suffix Array Language Model

- Allows for arbitrarily long history
- $P(e_i | e_{i-n+1}^{i-1})$
- Search for frequencies of all the embedded $k$-grams in $n$-gram $e_{i-n+1}^i$
- Discounting using Good-Turing smoothing with count-of-counts information collected from suffix array scanning
- Estimate probability of $e_i$ given histories of different length
- Interpolate $P(e_i | e_{i-k+1}^{i-1})$
Calling Suffix Array Language Model

- Index the training corpus
- Create a suffix array language model object: `C_SuffixArrayLanguageModel salm(corpus name)`
- When a new sentence starts: `lmState = salm.beginOfSentenceState();`
- Estimate the next word given current history: `logProb+=salm.logProb(lmState, nextWord, nextState);`
- Update the history with the next word: `lmState = nextState`
- At the end of the sentence: `logProb+=salm.logProbEnd(lmState)`
Discussions

- Distributed SALM using client/server architecture can handle arbitrarily large corpus
- Statistics of distant $n$-grams can be obtained through the operations of intersections of sentence IDs
- SALM does not support fuzzy match
- SALM download: http://projectile.is.cs.cmu.edu/research/public/tools/salm/salm.htm
- You are encouraged to use SALM to develop applications for your work