Basic Segmentation Methods

Richard Wicentowski

Computer Science Department
Swarthmore College

22-26 August 2005
Outline

1. Harris (1955) and Harris (1967)
Zellig Harris (1955) “From Phoneme To Morpheme”
Zellig Harris (1967) “Morpheme Boundaries Within Words: Report on a Computer Test”

- Harris proposes the use of successor frequency to find word and morpheme boundaries in phoneme utterances.
- The utterances are not separated into words.
- Input: /hiyzkwikər/ (He’s quicker)
- Output: /hiy.z.kwik.ər/

It is more natural to explain in terms of segmenting space-separated words rather than phoneme transcriptions, so we will avoid Harris’ examples.
Successor Counts

Given the first \( n \) letters of a word, \( W = w_1w_2...w_k \), the successor frequency at position \( n + 1 \) is defined to be the number of unique letters which follow \( w_1w_2...w_n \) is a large corpus.

- The basic procedure is to compute the successor frequency at every position of \( W \)
- Segment \( W \) where the number of successors reaches a peak or plateau
- The idea is that inside a word unit, the choices of successor are more limited, but at the boundaries of two units, we have a much less restricted choice.
Successor Counts: An example

Counts based on the most frequent 25,000 words length 3 or more from the all lower-case BNC corpus.

Find the successor count at each position of adaptations:

25
adaptations

Example words beginning a

abandon  academy  adapt  aerial  after  again  ahead
aid  akin  alarm  amass  and  apart  aquatic
...

25 possible next letters: abcdefghiklmnopqrstuvwxyz
Successor Counts: An example

Counts based on the most frequent 25,000 words length 3 or more from the all lower-case BNC corpus.

Find the successor count at each position of adaptations:

25 12
adaptations

Example words beginning ad
adamant add adept adhere adidas adjourn admiral adobe adrift ads adult advance

12 possible next letters: adehijmorsuv
Successor Counts: An example

Counts based on the most frequent 25,000 words length 3 or more from the all lower-case BNC corpus.

Find the successor count at each position of *adaptations*:

```
 25  12  3
adaptations
```

Example words beginning *ada*

```
adamanat  adapt  ada#
```

3 possible next letters: *mp#*

(# is the end of word symbol)
Successor Counts: An example

Counts based on the most frequent 25,000 words length 3 or more from the all lower-case BNC corpus.

Find the successor count at each position of adaptations:

25 12 3 1
adaptations

Example words beginning adap
adapt

1 possible next letter: p
Successor Counts: An example

Counts based on the most frequent 25,000 words length 3 or more from the all lower-case BNC corpus.

Find the successor count at each position of adaptations:

25 12 3 1 5
adaptations

Example words beginning adapt

adaptable adapted adapting adaptor adapt#

5 possible next letters: aeio#
Successor Counts: An example

Counts based on the most frequent 25,000 words length 3 or more from the all lower-case BNC corpus.

Find the successor count at each position of adaptations:

\[
\begin{array}{cccccc}
25 & 12 & 3 & 1 & 5 & 2 \\
adaptations
\end{array}
\]

Example words beginning adapta

<table>
<thead>
<tr>
<th>adaptable</th>
<th>adaptation</th>
</tr>
</thead>
</table>

2 possible next letters: bt
Successor Counts: An example

Counts based on the most frequent 25,000 words length 3 or more from the all lower-case BNC corpus.

Find the successor count at each position of *adaptations*:

```
25 12 3 1 5 2 1 1 1 1 2 1
adaptations
```

... and so on ...
The basic procedure

Now we segment at positions where there is a peak (or plateau) in the successor count.

```
25 12 3 1 5 2 1 1 1 1 2 1
adaptations
```
The basic procedure

Now we segment at positions where there is a peak (or plateau) in the successor count.

```
26 25 12 3 1 5 2 1 1 1 2 1
#  a  d  a  p  t  a  t  i  o  n  s
↑
```

We do not segment after the first letter a. Since there are 26 possible letters to start a word with, we are not really at a peak.

Of course, this means it will rarely be possible to split after the first letter, so one letter prefixes (e.g. English’s a: amoral, asymptomatic, etc), will fail to be detected.
The basic procedure

Now we segment at positions where there is a peak (or plateau) in the successor count.

```
2 6 2 5 1 2 3 1 5 2 1 1 1 2 1 0
# a d a p t a t i o n s #
```

Note that there is a symmetric end-of-character symbol, but it always has a successor count of 0.
The basic procedure

Now we segment at positions where there is a peak (or plateau) in the successor count.

```
26 25 12 3 1 5 2 1 1 1 2 1 0
# adaptations #
↑↑
```

We segment after the letter \( t \) and after the letter \( n \):

```
adapt - ation - s
```
Counting the counts

- How should keep track of all the successor frequencies?
- If this sounds to you like a computer science question, you’re right.
- We will store these counts in a prefix trie.
- If this was the only place you’d see tries this week, we wouldn’t be talking about it.
- Tries are a compact way of storing a large dictionary.
- As an example, a ≈ 235K-entry word list containing 2, 242, 389 individual letters was stored in a trie using only 756, 242 letters.
The trie data structure
Problem with successor counts

- The successor count idea makes some good choices, but also makes lots of bad choices.
- The successor count at position 2 of words beginning “de-” indicates we should make a split.
- This is correct for only about 1 in 3 words.
- Some examples of the bad split choice:

  deacon  dead  deaf  deal  dealt  dean
  dear    death  debacle  debate  debauchery  debit
  debris  debt  decade  dedicate  deduce  deduct
  deep    defeat  defend  defer  deficit  deficiency
Problems with successor counts

- Doesn’t take into account the differences in successor counts following consonants vs. vowels.
- Doesn’t segment properly when a lot of suffixes begin with the same letter:

<table>
<thead>
<tr>
<th>Correct</th>
<th>SC Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>chant-er</td>
<td>chante-r</td>
</tr>
<tr>
<td>chant-e</td>
<td>chante-</td>
</tr>
<tr>
<td>chant-es</td>
<td>chante-s</td>
</tr>
<tr>
<td>chant-ez</td>
<td>chante-z</td>
</tr>
<tr>
<td>chant-ent</td>
<td>chante-nt</td>
</tr>
</tbody>
</table>
Problems with successor counts

Near the end of long words, the successor counts become small because few other words begin with the same letters.

Successor Counts →

12 23 10 7 3 2 2 1 1 1 1 1 1
recognition

Harris proposes the Predecessor Count
We compute predecessor counts in much the same way.

Successor Counts →

<table>
<thead>
<tr>
<th>26</th>
<th>25</th>
<th>12</th>
<th>3</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>p</td>
<td>t</td>
<td>a</td>
<td>t</td>
<td>i</td>
<td>o</td>
<td>n</td>
<td>s</td>
<td>#</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>16</td>
<td>9</td>
<td>9</td>
<td>18</td>
<td>11</td>
<td>23</td>
<td>25</td>
</tr>
</tbody>
</table>

← Predecessor Counts

As before, we can also compute the successor and predecessor counts at the beginning and end of the word.

Splitting only using the predecessor count peaks yields

adapt - ati - ons
Returning to our previous example...

Successor Counts →

12 23 10 7 3 2 2 1 1 1 1 1 1 1
reconstitute
derede

← Predecessor Counts

Notice that now we have a peak at the end of the word which indicates we should be segmenting just before the e.
Harris also proposes to examine the average successor count of all letters two places ahead.

He calls this the \((n + 2)^{th}\) successor count, calling the previously defined successor count the \((n + 1)^{th}\).

The \((n + 2)^{th}\) successor frequency is the number of unique letter bigrams following \(w_1 \ldots w_n\).

The average is just the \((n + 2)^{th}\) successor count divided by the \((n + 1)^{th}\) successor count.
Rather than splitting at the first character after a peak in the $(n + 2)^{th}$ successor count, we split at the second character after a peak.
The \((n + 2)^{th}\) successor count

<table>
<thead>
<tr>
<th>Average ((n + 2)^{th}) Successor Counts</th>
<th>((n + 1)^{th}) Successor Counts</th>
<th>Predecessor Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 8 3 1 5 1 2 1 1 2 1 0 0</td>
<td>26 25 12 3 1 5 2 1 1 1 2 1 0</td>
<td>0 1 1 1 2 9 16 9 9 18 11 23 25</td>
</tr>
</tbody>
</table>

Splitting only using the \((n + 2)^{th}\) peak yields

adap - at - ions
Hafer and Weiss (1974)


- Hafer and Weiss extend the work of Harris
- Want to find not only a segmentation, but also to choose the stem
- Apply the output of their system to an IR task
- Get virtually identical results on IR task with hand stemming vs. their segmentation method
Four basic strategies

Hafer and Weiss propose four basic strategies for segmenting:

1. **Cutoff**: Pick a cutoff $K$ and segment when the successor count (or predecessor or both) reaches $K$: $SF(i) \geq K$

2. **Peak and Plateau**: Harris’ basic strategy. Cut whenever the current count is greater than or equal to the previous and following counts: $SF(i) \geq SF(i - 1)$ and $SF(i) \geq SF(i + 1)$

3. **Complete Word**: Segment whenever a word prefix/suffix matches a complete word in the corpus.

4. **Entropy**: Segment whenever the successor/predecessor entropy reaches a cutoff $K$. 
Entropy measures the number of bits that are required to compress information an the optimal coding scheme.

We compute entropy over a random variable $X$ as follows:

$$H(X) = - \sum_{x \in X} p(x) \log_2 p(x)$$

The following example is from Jurafsky and Martin (2000)
Entrophy: An example

- There are 8 horses, numbered 1 through 8, at a horse race.
- We transmit the horse we want to bet on to our agent.
- We can transmit using the following code:

<table>
<thead>
<tr>
<th>Horse 1</th>
<th>001</th>
<th>Horse 5</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse 2</td>
<td>100</td>
<td>Horse 6</td>
<td>110</td>
</tr>
<tr>
<td>Horse 3</td>
<td>011</td>
<td>Horse 7</td>
<td>111</td>
</tr>
<tr>
<td>Horse 4</td>
<td>100</td>
<td>Horse 8</td>
<td>000</td>
</tr>
</tbody>
</table>

- Each transmission requires 3 bits.
- We might be able to do better
Entropy: An example

- What if you knew Horse 1 won half the time and that Horses 5-8 hardly ever won?
- You could reduce the number of bits used to transmit Horse 1 and increase the bits used to transmit Horses 5-8.
- Your average number of transmitted bits per race would go down.
Entropy: An example

Here are the prior probabilities of each horse winning:

<table>
<thead>
<tr>
<th>Horse</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse 1</td>
<td>1/2</td>
</tr>
<tr>
<td>Horse 2</td>
<td>1/4</td>
</tr>
<tr>
<td>Horse 3</td>
<td>1/8</td>
</tr>
<tr>
<td>Horse 4</td>
<td>1/16</td>
</tr>
<tr>
<td>Horse 5</td>
<td>1/64</td>
</tr>
<tr>
<td>Horse 6</td>
<td>1/64</td>
</tr>
<tr>
<td>Horse 7</td>
<td>1/64</td>
</tr>
<tr>
<td>Horse 8</td>
<td>1/64</td>
</tr>
</tbody>
</table>

The entropy of the random variable $X$ over horses is:

$$H(X) = - \sum_{i=1}^{8} p(i) \log_2 p(i)$$

$$= - \left( \frac{1}{2} \log \frac{1}{2} + \frac{1}{4} \log \frac{1}{4} + \frac{1}{8} \log \frac{1}{8} + \frac{1}{16} \log \frac{1}{16} + 4 \left( \frac{1}{64} \log \frac{1}{64} \right) \right)$$

$$= 2 \text{ bits}$$
Entropy: An example

One possible coding scheme we could use that would transmit an average of 2 bits per race would be:

<table>
<thead>
<tr>
<th>Horse 1</th>
<th>0</th>
<th>Horse 5</th>
<th>111100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse 2</td>
<td>10</td>
<td>Horse 6</td>
<td>111101</td>
</tr>
<tr>
<td>Horse 3</td>
<td>110</td>
<td>Horse 7</td>
<td>111110</td>
</tr>
<tr>
<td>Horse 4</td>
<td>1110</td>
<td>Horse 8</td>
<td>111111</td>
</tr>
</tbody>
</table>
Entropy: An example

What if each horse was equally likely?

<table>
<thead>
<tr>
<th>Horse 1</th>
<th>Horse 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{8}$</td>
<td>$\frac{1}{8}$</td>
</tr>
<tr>
<td>Horse 2</td>
<td>Horse 6</td>
</tr>
<tr>
<td>$\frac{1}{8}$</td>
<td>$\frac{1}{8}$</td>
</tr>
<tr>
<td>Horse 3</td>
<td>Horse 7</td>
</tr>
<tr>
<td>$\frac{1}{8}$</td>
<td>$\frac{1}{8}$</td>
</tr>
<tr>
<td>Horse 4</td>
<td>Horse 8</td>
</tr>
<tr>
<td>$\frac{1}{8}$</td>
<td>$\frac{1}{8}$</td>
</tr>
</tbody>
</table>

The entropy of the random variable $X$ over horses is:

$$H(X) = - \sum_{i=1}^{8} p(i) \log_2 p(i)$$

$$= - 8 \times \frac{1}{8} \log_2 \frac{1}{8}$$

$$= 3 \text{ bits}$$
Successor Entropy

How does successor entropy work and why is it useful?

- Assume that at position $i$ of two words, $W_1$ and $W_2$, both have successor counts of 10.
- For simplicity assume the successors are the letters $a$ through $j$.
- Assume that the first $i$ letters of $W_1$ match 100 corpus entries and that each of the 10 successor letters occurs in 10 words.
- Assume that the first $i$ letters of $W_2$ match 19 corpus entries. Of these, 10 have $a$ as the next letter, and 1 each have $b$ through $j$.
- It seems like we’d want to segment $W_1$, but maybe not $W_2$. 
Successor Entropy

The successor entropy at position $i$ of $W_1$ is:

$$H(X) = - \sum_{i=1}^{10} p(i) \log_2 p(i)$$

$$= - 10 \times \frac{1}{10} \log \frac{1}{10}$$

$$= 3.32 \text{ bits}$$

The successor entropy at position $i$ of $W_2$ is:

$$H(X) = - \sum_{i=1}^{10} p(i) \log_2 p(i)$$

$$= - \frac{10}{19} \log \frac{10}{19} + 9 \times \frac{1}{19} \log \frac{1}{19}$$

$$= 2.50 \text{ bits}$$
Successor Entropy: An example

<table>
<thead>
<tr>
<th>26</th>
<th>25</th>
<th>12</th>
<th>3</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Successor Counts $\rightarrow$

<table>
<thead>
<tr>
<th>4.3</th>
<th>3.9</th>
<th>3.0</th>
<th>1.1</th>
<th>0.0</th>
<th>2.1</th>
<th>1.0</th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>1.0</th>
<th>0.0</th>
<th>0.0</th>
</tr>
</thead>
</table>

Successor Entropy $\rightarrow$

#adaptations#

← Predecessor Entropy

<table>
<thead>
<tr>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>1.0</th>
<th>2.9</th>
<th>3.5</th>
<th>2.2</th>
<th>1.1</th>
<th>1.3</th>
<th>1.9</th>
<th>3.5</th>
<th>3.6</th>
</tr>
</thead>
</table>

← Predecessor Counts

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>9</th>
<th>16</th>
<th>9</th>
<th>9</th>
<th>18</th>
<th>11</th>
<th>23</th>
<th>25</th>
</tr>
</thead>
</table>

Splitting when the predecessor entropy exceeds 3.0 (used in Hafer and Weiss experiments) yields

adapt - ation - s
Stem determination

Hafer and Weiss declare the first segment to be the stem unless:

1. If the first segment appears in more than $K$ words, it is probably a prefix, so choose the second segment. They found $K = 12$ worked best.

2. If the first and the second segment are both words in the corpus, e.g. school–child then both are considered stems.

In their results section, that they found that being conservative in the splitting (and hence, higher precision, lower recall) gives much more desirable results.
### Segmentation Results (from Hafer & Weiss)

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>.894</td>
<td>.511</td>
<td>.650</td>
</tr>
<tr>
<td>3</td>
<td>.848</td>
<td>.565</td>
<td>.678</td>
</tr>
<tr>
<td>4</td>
<td>.904</td>
<td>.318</td>
<td>.470</td>
</tr>
<tr>
<td>6</td>
<td>.778</td>
<td>.711</td>
<td>.743</td>
</tr>
<tr>
<td>7</td>
<td>.486</td>
<td>.734</td>
<td>.585</td>
</tr>
<tr>
<td>8</td>
<td>.787</td>
<td>.569</td>
<td>.660</td>
</tr>
<tr>
<td>9</td>
<td>.441</td>
<td>.828</td>
<td>.575</td>
</tr>
<tr>
<td>10</td>
<td>.484</td>
<td>.937</td>
<td>.638</td>
</tr>
<tr>
<td>11</td>
<td>.910</td>
<td>.610</td>
<td>.730</td>
</tr>
<tr>
<td>12</td>
<td>.720</td>
<td>.728</td>
<td>.724</td>
</tr>
<tr>
<td>13</td>
<td>.609</td>
<td>.596</td>
<td>.602</td>
</tr>
<tr>
<td>14</td>
<td>.874</td>
<td>.526</td>
<td>.657</td>
</tr>
<tr>
<td>15</td>
<td>.818</td>
<td>.700</td>
<td>.754</td>
</tr>
</tbody>
</table>

No results are provided for experiments 1 and 5.
Stemming Results

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Precision</th>
<th>Experiment</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0%</td>
<td>9</td>
<td>19.7%</td>
</tr>
<tr>
<td>2</td>
<td>17.9%</td>
<td>10</td>
<td>11.7%</td>
</tr>
<tr>
<td>3</td>
<td>20.4%</td>
<td>11</td>
<td>35.3%</td>
</tr>
<tr>
<td>4</td>
<td>0.0%</td>
<td>12</td>
<td>59.4%</td>
</tr>
<tr>
<td>5</td>
<td>0.0%</td>
<td>13</td>
<td>2.6%</td>
</tr>
<tr>
<td>6</td>
<td>16.5%</td>
<td>14</td>
<td>35.4%</td>
</tr>
<tr>
<td>7</td>
<td>17.8%</td>
<td>15</td>
<td>22.8%</td>
</tr>
<tr>
<td>8</td>
<td>25.0%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These are results that I got testing on English verbs.
Hervé DéJean (1998) “Morphemes As Necessary Concept For Structures Discovery from Untagged Corpora”

- Start with a 100K word corpus
- Segment if successor count is greater than a threshold (where threshold is set to half the number of letters in the alphabet)
- Do not segment if we can find a longer morpheme to segment (e.g. don’t segment -on if we can segment -ion)
- Keep only those sequences which occur at least 100 times and call these morphemes.
Do not segment if we can find a longer morpheme to segment (e.g. don’t segment $-on$ if we can segment $-ion$)...

1. Count the number of words (types) ending in $-on$ (367).
2. Count the number of words that end in $-ion$ (292).
3. If the count of $-ion$ is more than 50% of the count of $-on$ (here, 80%), then consider the sequence $-on$ to be part of the larger sequence $-ion$. 
Iteration

- For each letter sequence (e.g. light), consider the letters which follow.
- If at least half are morphemes we have already found (-s, -ed, -ing, -ly, -er), then we consider the remainder (-ness, -en, -est) as morphemes.
- Since this process generates incorrect morphemes, keep only these new morphemes which occur more than 5 times.
DéJean gives few results, citing only that a hand-inspected sample of 500 words yielded 8 improper segmentations. Does not say how many of these 500 were segmented. Does not say how many should have been segmented but were not. May be a good (and easy) experiment to duplicate.
While none of the experiments gets superb results, they are certainly reasonable enough to use as part of a bootstrapping process.