T-(538|725)-MALV, Natural Language Processing
Word counting and n-grams

Hrafn Loftsson\textsuperscript{1}  Hannes Högni Vilhjálmsson\textsuperscript{1}

\textsuperscript{1}School of Computer Science, Reykjavik University

September 2008
1. Word sequences

2. The construction of n-gram language models

3. Probabilistic models

4. Smoothing
Outline

1 Word sequences

2 The construction of n-gram language models

3 Probabilistic models

4 Smoothing
Collocations

- A sequence of words or terms which co-occur more often than would be expected by chance

- Phrases composed of words that co-occur for lexical rather than semantic reasons
  - “Heavy smoker” vs. “heavy writer”

- Often it is important to find collocations, for example in the construction of dictionaries.

- Examples of collocations: “crystal clear”, “cosmetic surgery”, “blonde hair”, “oft og tíðum”, “veikur hlekkur”.
Language models

Language model (í. Mállíkan)

- A probabilistic estimation for the frequency of words and word sequences.
- Often used to predict the next word when the preceding sequence is known.
- Used in many NLP applications:
  - Speech recognition, POS tagging, parsing, semantic analysis, machine translation, etc.
# Word types and tokens

<table>
<thead>
<tr>
<th>Word types (í. Orðmyndir)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distinct words.</td>
</tr>
<tr>
<td>The <em>IFD</em> corpus contains 59,358 word types.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word tokens (í. Tókar/lesmálsorð)</th>
</tr>
</thead>
<tbody>
<tr>
<td>All words/tokens.</td>
</tr>
<tr>
<td>The <em>IFD</em> corpus contains 590,297 tokens.</td>
</tr>
</tbody>
</table>

## An example

- Þetta er skóli. Anna sá skóla. Atli sá skóla.
- 12 tokens, 8 word types.
Word types and tokens

**Word types (í. Orðmyndir)**

- Distinct words.
- The *IFD* corpus contains 59,358 word types.

**Word tokens (í. Tókar/lesmálsorð)**

- All words/tokens.
- The *IFD* corpus contains 590,297 tokens.

**An example**

- Þetta er skóli. Anna sá skóla. Atli sá skóla.
- 12 tokens, 8 word types.
n-grams (í. N-stæður)

- A sequence of $N$ words/tokens.
- Unigrams (í. Einstæður)
- Bigrams (í. Tvístæður)
- Trigrams (í. Prístæður)
- Fourgrams (í. Fjórstæður).
- etc.
n-grams

- Þetta er skóli. Anna sá skóla. Atli sá skóla.
- Bigrams: “Þetta er”, “er skóli”, “skóli .”, “. Anna”, “Anna sá”, “sá skóla”, etc.
- Trigrams: “Þetta er skóli”, “er skóli .”, “skóli . Anna”, “. Anna sá”, “Anna sá skóla”, “sá skóla .”, etc.
1. Word sequences

2. The construction of n-gram language models

3. Probabilistic models

4. Smoothing
More Unix/Linux tools

**sort**

- **Alphabetical order:**
  - `sort inputfile > outputfile` (ascending order)
  - Sorting Icelandic text works under Linux using UTF-8 file encoding for data
  - Does it work in Cygwin?

- **Descending order:**
  - `sort -r inputfile > outputfile`

- **Numerical sort**
  - `sort -n inputfile > outputfile`
More Unix/Linux tools

### uniq

- Eliminates or counts duplicate lines in a presorted file
- `uniq inputfile > outputfile`
- `sort input.txt | uniq > output.txt`
- With frequency:
  - `uniq -c inputfile > outputfile`
- Counting frequencies
  - `sort input.txt | uniq -c | sort -nr > output.txt`
The construction of n-gram language models

A unigram model

- **Input:** A corpus.
  1. **Tokenisation** – one word (token) per line.
  2. **Counting.**

Easy in Unix/Linux

- Let us assume that the file *corpus.wrd* contains one token per line.
- `sort corpus.wrd | uniq -c | sort -nr > corpus.freq`
The construction of n-gram language models

A unigram model

- Input: A corpus.
  1. Tokenisation – one word (token) per line.
  2. Counting.

Easy in Unix/Linux

- Let us assume that the file `corpus.wrd` contains one token per line.
- `sort corpus.wrd | uniq -c | sort -nr > corpus.freq`
use utf8; # allow UTF-8 in the program text
$file = shift(@ARGV); # get the input file name
$outfile = shift(@ARGV); # get the output file name
open(INFILE, "<:utf8", "$file"); # open the file using utf8 encoding
open(OUTFILE, ">:utf8", "$outfile"); # write using utf8 encoding

$text = <INFILE>;
while ($line = <INFILE>) { $text .= $line }
$text =~ tr /a-zA-Záðéíóýúæöþ0-9.,()!?-:;/
/cs; # The not so perfect tokenisation step
$text =~ s/\([,.?!:;()-\ ]/\n$1/\g;
$text =~ s/\n+/\n/\g;

@words = split(/\n/, $text);
for ($i=0; $i <= $#words; $i++) {
    if (!exists($frequency{$words[$i]})) {$frequency{$words[$i]} = 1;}
    else {$frequency{$words[$i]}++;}
}

foreach $word (sort keys %frequency) {
    print OUTFILE "$frequency{$word} $word\n";
}
More Unix/Linux tools

**head og tail**

- head -3 < input.txt
  - Returns the first three lines.
- tail -2 < input.txt
  - Returns the last two lines.
- tail +2 < input.txt
  - Skips the first line.
- If this does not work, then tail --lines=+2 < input.txt
  - Skips the first line.
The construction of n-gram language models

A bigram model

- Input: A corpus.
  1. Tokenisation – one word (token) per line.
  2. Construct bigrams: Print out $word_i$ and $word_{i+1}$ in the same line.
  3. Counting.

Easy in Unix/Linux

- Let us assume that the file corpus.wrd contains one token per line.
- `tail --lines=+2 < corpus.wrd > corpus2.wrd`
- `paste corpus.wrd corpus2.wrd > corpus.bigrams`
- `sort corpus.bigrams | uniq -c | sort -nr > corpus.freq`
## The construction of n-gram language models

### A bigram model

- **Input:** A corpus.
  1. Tokenisation – one word (token) per line.
  2. Construct bigrams: Print out $word_i$ and $word_{i+1}$ in the same line.
  3. Counting.

### Easy in Unix/Linux

- Let us assume that the file `corpus.wrd` contains one token per line.
- `tail --lines=+2 < corpus.wrd > corpus2.wrd`
- `paste corpus.wrd corpus2.wrd > corpus.bigrams`
- `sort corpus.bigrams | uniq -c | sort -nr > corpus.freq`
use utf8; # allow UTF-8 in the program text
$file = shift(@ARGV); $outfile = shift(@ARGV); # get the output file name
open(INFILE, "<:utf8", "$file"); open(OUTFILE, ">:utf8", "$outfile");

$text = <INFILE>;
while ($line = <INFILE>) { $text .= $line}
$text =~ tr /a-záðéíóúýúæöþA-ZÁÐÉÍÓÝÚÆÖÝ0-9().,!?:;/'
/cs;
$text =~ s/([^,?!:;()\-])/\n$1
/g;
$text =~ s/\n+/\n/g;
@words = split(/\n/, $text);

for ($i=0; $i<$#words; $i++) {
    $bigrams[$i] = $words[$i] . " " . $words[$i+1]; }

for ($i=0; $i <= $#bigrams; $i++) {
    if (!exists($frequency{$bigrams[$i]})) {$frequency{$bigrams[$i]} = 1;}
    else {$frequency{$bigrams[$i]}++;
}
foreach $bigram (sort keys %frequency) {
    print OUTFILE "$frequency{$bigram} $bigram\n";
}
Outline

1. Word sequences
2. The construction of n-gram language models
3. Probabilistic models
4. Smoothing
Probabilistic models of a word sequence

Maximum likelihood estimation (í. Sennileikalíkur)

- Let $S = w_1, w_2, \ldots, w_n$ be a word sequence.
- By using a (training) corpus $M$, we can estimate the probability of this sequence.
- $P(S)$ is the relative frequency of the string $S$ in $M$.
- $P(S)$ is called the *maximum likelihood estimate* (MLE) for $S$:

$$P_{MLE}(S) = \frac{C(w_1, w_2, \ldots, w_n)}{N} \quad (1)$$

$N$ is the total number of strings of length $n$ in $M$. 
Probabilistic models of a word sequence

Maximum likelihood estimation

- Most of the time, it is impossible to obtain this estimate, because the size of a corpus is finite!
- We thus simplify (1) and decompose it:

\[
P(S) = P(w_1, w_2, \ldots, w_n)
= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \ldots P(w_n|w_1, \ldots, w_{n-1})
= \prod_{i=1}^{n} P(w_i|w_1, \ldots, w_{i-1})
\]
Probabilistic models of a word sequence

The length of the n-grams needs to be limited

- \( P(\text{It was a bright cold day in April}) \)
- \( P(S) = P(\text{It}) * P(\text{was}|\text{It}) * P(\text{a}|\text{It, was}) * P(\text{bright}|\text{It, was, a}) \ldots * P(\text{April}|\text{It, was, a, bright \ldots, in}) \)
- In this example, we even need 8-gram statistics. No corpus is big enough to produce them. We thus approximate these probabilities (using the Markov assumption) with bigrams or trigrams:

\[
P(w_i|w_1, \ldots, w_{i-1}) \approx P(w_i|w_{i-1}) \tag{2}
\]

\[
P(w_i|w_1, \ldots, w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1}) \tag{3}
\]
Probabilistic models of a word sequence

The probability of a sentence using bigrams and trigrams

\[ P(S) = P(w_1) \prod_{i=2}^{n} P(w_i|w_{i-1}) \]

\[ P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})} \]

\[ P(S) = P(w_1)P(w_2|w_1) \prod_{i=3}^{n} P(w_i|w_{i-2}, w_{i-1}) \]

\[ P_{MLE}(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})} \]
Training, development and testing

Different kind of usage

- **Training corpus** (í. Þjálfunarmálheild):
  - A corpus used to derive the n-gram frequencies (the language model).
- **Test corpus** (í. Prófunarmálheild):
  - The corpus on which we apply the model.
- **Development corpus** (í. Þróunarmálheild):
  - A corpus used to fine-tune some parameters used by the model.
- All the three corpora need to be distinct.
\textit{n–fold cross-validation}

- A corpus divided randomly into two parts, a training corpus and a test corpus.
- The language model trained using the training corpus and the model applied on the test corpus.
- Repeated \(n\)-times, each time with a new random division.
- Results are averaged.
- Called \(10–fold \text{ cross-validation}\) when \(n = 10\).
- Results are not dependent on one specific division between training and test sets.
Words, which are not part of the language model (i.e. have not been encountered during training), will appear during testing. Why is that almost certain? These words are called unknown or out-of-vocabulary (OOV) words. Moreover, the estimated frequency of unknown words is not very reliable.

Two approaches for handling unknown words:
- *Closed vocabulary*. Unknown words discarded.
- *Open vocabulary*. Unknown words handled in a specific manner, e.g. using smoothing.
Language models are derived from corpora which are not large enough to produce reliable frequencies for all possible bigrams and trigrams.

Given a vocabulary of 20,000 word types:
- Bigrams: $20,000^2 = 400,000,000$
- Trigrams: $20,000^3 = 8,000,000,000,000$

Training data is thus sparse. Many n-grams will get the probability 0, which is not realistic (see an example on page 99).

The MLE method gives no hint how to estimate probabilities for unseen n-grams.

⇒ Smoothing (í. Sléttun)
Outline

1. Word sequences
2. The construction of n-gram language models
3. Probabilistic models
4. Smoothing
Smoothing

Laplace’s Rule (1820)

- Simply adds one to all frequencies.
- ⇒ “the add one method”.
- The frequency of unseen n-grams is thus 1.

\[ P_{Laplace}(w_{i+1}|w_i) = \frac{C(w_i, w_{i+1}) + 1}{C(w_i) + \text{Card}(V)} \]

\text{Card}(V) is the number of word types.
### Table: Frequencies of bigrams using Laplace’s rule

| $w_i, w_{i+1}$ | $C(w_i, w_{i+1})$ | $C(w_i) + \text{Card}(V)$ | $P_{\text{Lap}}(w_{i+1} | w_i)$ |
|----------------|-------------------|-----------------|-----------------|
| &lt;s&gt;a     | 133               | 7072 + 8634     | 0.008500        |
| a good         | 14                | 2482 + 8634     | 0.001300        |
| good deal      | 0                 | 53 + 8634       | 0.000120        |
| deal of        | 1                 | 5 + 8634        | 0.000230        |
| of the         | 742               | 3310 + 8634     | 0.062000        |
| the literature | 1                 | 6248 + 8634     | 0.000130        |
| literature of  | 3                 | 7 + 8634        | 0.000460        |
| of the         | 742               | 3310 + 8634     | 0.062000        |
| the past       | 70                | 6248 + 8634     | 0.004800        |
| past was       | 4                 | 99 + 8634       | 0.000570        |
| was indeed     | 0                 | 2211 + 8634     | 0.000092        |
| indeed already | 0                 | 17 + 8634       | 0.000120        |
| already being  | 0                 | 64 + 8634       | 0.000110        |
| ...            |                   |                 |                 |
| this way       | 3                 | 264 + 8634      | 0.000450        |
Smoothing

Drawback

- Unseen n-grams receive an enormous mass of probabilities
  - The unlikely bigram the of gets the frequency 1, one fourth of the frequency of the (common) bigram this way.

- Discount factor is the ratio between the MLE frequencies and the smoothed frequencies. This factor is often too high when Laplace’s rule is used.

- Example:
  - According to the language model, the MLE probability for this way is $\frac{3}{264} = 0.0114$. After smoothing, the probability is 0.00045.
  - Discount factor is: $\frac{0.0114}{0.00045} = 24.4$.
  - The MLE probability for this bigram has been discounted by 24.4 (to make place for the unseen bigrams).
Good-Turing estimation (1953)

- One of the most efficient smoothing methods.
- It reestimates the counts of n-grams observed in the corpus by discounting them, and shifts probability mass it has shaved to the unseen bigrams (as Laplace’s rule).
- However, *Discount factor* is variable, and depends on the number of times a n-gram has occurred in the corpus.

Definition

- Let $N_c$ be the number of n-grams that occurred exactly $c$ times in the corpus.
- $N_0$ is the number of unseen n-grams, $N_1$ is the number of n-grams seen once, etc.
Good-Turing estimation (1953)

- One of the most efficient smoothing methods.
- It reestimates the counts of n-grams observed in the corpus by discounting them, and shifts probability mass it has shaved to the unseen bigrams (as Laplace’s rule).
- However, *Discount factor* is variable, and depends on the number of times a n-gram has occurred in the corpus.

Definition

- Let $N_c$ be the number of n-grams that occurred exactly $c$ times in the corpus.
- $N_0$ is the number of unseen n-grams, $N_1$ is the number of n-grams seen once, etc.
Good-Turing

- Reestimates the frequency of n-grams occurring $c$ times, with the formula:

$$c^* = (c + 1) \frac{N_{c+1}}{N_c}$$

- For unseen n-grams: $c^* = \frac{N_1}{N_0}$

- For n-grams occurring once: $c^* = \frac{2N_2}{N_1}$

- For n-grams occurring twice: $c^* = \frac{3N_3}{N_2}$

- The conditional frequency is:

$$P_{GT}(w_n | w_1, \ldots, w_{n-1}) = \frac{c^*(w_1, \ldots, w_n)}{C(w_1, \ldots, w_{n-1})}$$
Table: The reestimated frequencies of the bigrams using Good-Turing smoothing.

Note that the reestimated frequency of bigrams not seen during training is only 0.0005.
Linear interpolation

- Linear interpolation = Deleted interpolation
- Combines linearly the MLE of length 1 to n.
- The estimation of each unseen n-gram depends on the exact words comprising the n-gram.
- For trigrams:
  \[
  P_{\text{DelInterpolation}}(w_n|w_{n-2}, w_{n-1}) = \\
  \lambda_1 P_{\text{MLE}}(w_n|w_{n-2}, w_{n-1}) + \\
  \lambda_2 P_{\text{MLE}}(w_n|w_{n-1}) + \lambda_3 P_{\text{MLE}}(w_n)
  \]
  where \(0 \leq \lambda_i \leq 1\) and \(\sum_{i=1}^{3} \lambda_i = 1\)
- The \(\lambda_i\) can be trained and optimised from a corpus.