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

Hrafn Loftsson[1] Hannes Högni Vilhjálmsson[1]

[1]School of Computer Science, Reykjavik University

September 2009

Outline

Outline

Collocations

Collocations (í. Orðastæður)

- 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 preceeding sequence is known.
- Used in many NLP applications:
  - Speech recognition, PoS tagging, parsing, semantic analysis, machine translation, etc.

Word types and tokens

Word types (í. Orðmyndir)

- Distinct words.
- The Icelandic Frequency Dictionary (*IFD*) corpus contains 59,358 word types.

Word tokens (í. Tókar/lesmálsorð)

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

Word types and tokens

An example

- This is a school. Anna saw the school. John saw the school.
- 15 tokens, 12 word types.

n-grams (í. n-stæður)

- A sequence of N words (tokens).
- Unigrams (í. Einstæður)
- Bigrams (í. Tvístæður)
- Trigrams (í. Þrístæður)
- Fourgrams (í. Fjórstæður).
- etc.

n-grams

- This is a school.
- Bigrams: “This is”, “is a”, “a school”, “school .”
- Trigrams: “This is a”, “is a school”, “a school .”

Outline

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
- 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

Counting unigrams in Perl (cf. 4.4.3 in textbook)

```
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

while ($line = <INFILE>) { $text .= $line}
$text =~ tr /a-záðéíóýúæöþA-ZÁÐÉÍÓÝÚÆÖÞ0-9.,()!?\-:;/\n/cs; # The not so perfec
$text =~ s/([,.?!:;()\-])/\n$1\n/g;                           # tokenisation step
$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
- 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

A bigram model in Perl (cf. 4.4.4 in textbook)

```
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");

while ($line = <INFILE>) { $text .= $line}
$text =~ tr /a-záðéíóýúæöþA-ZÁÐÉÍÓÝÚÆÖÞ0-9().,!?\-:;/\n/cs;
$text =~ s/([,.?!:;()\-])/\n$1\n/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

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:

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

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(*It was a bright cold day in April*)
- P(S) = P(*It*) * P(*was*|*It*) * P(*a* | *It, was*) * P(*bright* | *It*, *was*, *a*) . . . * P(*April* | *It*, *was*, *a*, *bright* . . . , *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:

$$P(w_i|w_1,\ldots,w_{i-1}) \approx P(w_i|w_{i-1}) \quad (2)$$

$$P(w_i|w_1,\ldots,w_{i-1}) \approx P(w_i|w_{i-2},w_{i-1}) \quad (3)$$

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.

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 cross-validation* when $n = 10$.
- Results are not dependent on one specific division between training and test sets.

Vocabulary

- 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`.

Sparse data (í. Naum gögn)

- 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.
- $\Rightarrow$ Smoothing (í. Sléttun)

Outline

Smoothing

Laplace's Rule (1820)

- Simply adds one to all frequencies.
- $\Rightarrow$ "the add one method".
- The frequency of unseen n-grams is thus 1.

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

$Card(V)$ is the number of word types.

Smoothing – table page 100

w_i, w_{i+1}	$C(w_i, w_{i+1})$	$C(w_i) + Card(V)$	$P_{Lap}(w_{i+1} \mid w_i)$
<s>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

Table: Frequencies of bigrams using Laplace's rule

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).

Smoothing

Good-Turing estimation (1953)

- One of the most efficient smoothing methods.
- It reestimates the counts of n-grams observerd 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: $c* = \frac{2*N_2}{N_1}$
- For n-grams occurring twice: $c* = \frac{3*N_3}{N_2}$
- The conditional frequency is:

$$P_{GT}(w_n|w_1,\ldots,w_{n-1}) = \frac{c^*(w_1,\ldots,w_n)}{C(w_1,\ldots,w_{n-1})}$$

Good-Turing – table page 102

Frequency of occurrence	N_c	c*
0	74.523.701	0.0005
1	37.365	0.31
2	5.820	1.09
3	2.111	2.02
4	1.067	3.37
5	719	3.91
6	468	4.94
...		

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:
  $P_{DelInterpolation}(w_n|w_{n-2}, w_{n-1}) =$
  $\lambda_1 P_{MLE}(w_n|w_{n-2}, w_{n-1}) +$
  $\lambda_2 P_{MLE}(w_n|w_{n-1}) + \lambda_3 P_{MLE}(w_n)$
- where $0 \leq \lambda_i \leq 1$ and $\sum_{i=1}^{3} \lambda_i = 1$
- The λ_i can be trained and optimised from a corpus.