# Natural Language Processing

## Lab 3: Maximum Likelihood Estimation

---

## 1 Introduction

In this lab, we are going to start using statistical inference to build probability models from data. We will use an English text corpus to estimate bigram probabilities and then use probability theory to derive various related probabilities. Next time, we will explore the use of $n$-gram models for the problem of language modeling. All the files needed for the lab are in `/local/kurs/nlp/ngram/`. Make copies of the files into your home directory (as you did in Labs 1 and 2).

## 2 Data and preprocessing

Our statistical sample will be three collections of short stories about Sherlock Holmes by Arthur Conan Doyle,[1] available as plain text in the file `holmes.txt`. We will start by tokenizing the text and segmenting it into sentences. You could in principle use your own tokenizer, but to make sure that everyone has the same version of the text it is better to use `tokenizer5.py`, which uses the NLTK library. It is not perfect but it is good enough for our purposes.

## 3 Extract bigram frequencies

Estimation of probabilities is always based on frequency data, and we will start by computing the frequency of *word bigrams* in our corpus. This is what the Python program `bigrams.py` does.

```
import sys

start = "<s>"
cxt = start
word = ""
freq = {}
bigrams = {}
eos = False

# Count bigrams

for line in sys.stdin:
    if line == "\n":
        word = "<e>"
        eos = True
    else:
        word = line.rstrip()
    if cxt not in bigrams:
        bigrams[cxt] = []
    if (cxt, word) not in freq:
        freq[(cxt, word)] = 0
        bigrams[cxt].append(word)
    freq[(cxt,word)] += 1
    cxt = word
    if eos:
        cxt = start
        eos = False

# Print bigram frequencies

for c in sorted(bigrams.keys()):
    for w in sorted(bigrams[c]):
        print(c + "\t" + w + "\t" + str(freq[(c,w)]))
```

[1] The collections are *The Adventures of Sherlock Holmes*, *The Memoirs of Sherlock Holmes*, and *The Return of Sherlock Holmes*.

The program reads from `stdin` and prints to `stdout`, so in order to count the bigrams in the text you need to run something like:

```
python bigrams.py < holmes-tokens.txt > holmes-bigrams.txt
```

The output consists of all bigrams found in the text, sorted alphabetically with one bigram on each line, followed by its frequency.

**Note on sentence boundaries:** If you look closely at `bigrams.py` (or its output), you can see that the two special tokens `<s>` and `<e>` are inserted at sentence boundaries to mark the *start* and *end* of a sentence, respectively. This is necessary to get a properly normalized language model eventually, but for now you just need to be aware of it.

Have a look at the bigram frequencies derived from the Holmes corpus and try to answer the following questions before you go on to the next section:

1. How many bigram *tokens* are there in the corpus? How many bigram *types*?
2. What is the most frequent bigram in the corpus? What is the most frequent bigram that does not involve punctuation or one of the special tokens `<s>` and `<e>`?
3. What is the frequency of the bigram "Sherlock Holmes"? What about "Holmes Sherlock"?
4. How many bigrams only occur once?
5. How many bigrams do not occur at all (assuming that the entire vocabulary is made up of tokens that occur at least once in the corpus)?

**Note on linux commands:** The frequencies needed to answer these questions can be computed by a modified version of `bigrams.py`. In most cases, however, it is easier to just use standard unix/linux commands like `wc`, `grep`, and `sort`. If you are unsure how to do this, check out Appendix A.

## 4 Maximum likelihood estimation

The next step is to use the frequency counts to estimate a probability distribution over bigrams, that is, the joint distribution $P(w_1, w_2)$, where $w_1$ and $w_2$ are the first and second word of a bigram, respectively. We will use the maximum likelihood estimate (MLE). Start by answering the following questions:

1. What does it mean that an estimate is MLE?
2. Let $|W|$ be the number of words in our vocabulary, and let $P(w_1, w_2) = \frac{1}{|W|^2}$ for all $(w_1, w_2)$. Why is this not an MLE for our corpus?
3. What is the MLE of $P(\text{Sherlock}_1, \text{Holmes}_2)$ for our corpus?
4. What is the MLE of $P(\text{Holmes}_1, \text{Sherlock}_2)$?

When you have convinced yourselves that you can answer these questions, go ahead and compute the MLE for all word bigrams occurring in our corpus. For this you need to modify `bigrams.py` so that it prints the bigram probability estimate instead of the bigram frequency as the last element on each line .

## 5 More probability estimates

Given your estimates of the joint bigram probabilities, you should be able to derive estimates for the following:

1. The marginal probability $P(w_1)$
2. The conditional probability $P(w_2|w_1)$

Derive these estimates. (You don't need to implement this).

## A Useful unix/linux commands

- To count the number of lines, words, and characters in a file, use `wc`:

```
prompt$ wc holmes-tokens.txt
  132961  126113  587964 holmes-tokens.txt
```

  This tells us that `holmes-tokens.txt` consists of 132691 lines, 126113 words, 587964 characters.

- To sort the lines of a file, use `sort`. The flag `-k` can be used to specify which column should be used for sorting and the flag `-n` makes the sorting numerical rather than alphabetical. Thus, the following command sorts the bigram file with respect to frequency:

```
prompt$ sort -k 3 -n holmes-bigrams.txt
```

  The flag `-r` reverses the sort order (in this case to decreasing frequency instead of increasing).

- To find all lines that match a given regular expression, use `grep`. For example, this commands finds all the lines in the bigram file with a frequency of 10:

```
prompt$ grep '[^0-9]10$' holmes-bigrams.txt
```

  The flag `-v` inverts the search to match all lines **not** matching the pattern. Thus, the following command finds all non-empty lines in a file:

```
prompt$ grep -v '^$' holmes-bigrams.txt
```

- To count the number of unique lines (types rather than tokens), combine `sort`, `uniq` and `wc` as follows:

```
prompt$ sort holmes-tokens.txt | uniq | wc
    8807    8806   70763
```

  The `uniq` command removes adjacent repetitions of the same line, and the pipe symbol `|` is used to pass the output of one command as input to the next.