# Web Information Retrieval

**Assignment 5**

Lukas Schmelzeisen lukas@uni-koblenz.de
Dr. Chandan Kumar kumar@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Koblenz-Landau

Submission until: June 13, 2017, 10:00 a.m.
Tutorial on: June 13 and June 16, 2017

This assignment covers the *Language Models* in Information Retrieval. As supplementary material to the lecture, I can recommend reading *Chapter 12* of the book "Introduction to Information Retrieval" by Manning, C. D., Raghavan, P., and Schütze, H. from 2008. The book is available for free under http://informationretrieval.org

# 1 Query Likelihood Model (14 Points)

In the query likelihood model the likelihood of a document $d$ being relevant to a query $q$ is estimated as some quantity $P(d \mid q)$.

There are many choices on how to model this estimation. This question will explore two simple models:

1. A unsmoothed, unigram model:

$$P_{\text{uni}}(d \mid q) = \prod_{t \in q} P_{M_d}(t)$$

2. A linear-interpolated[1], unigram model ($0 \leq \lambda \leq 1$):

$$P_{\text{interp-uni}}(d \mid q) = \prod_{t \in q} \left[\lambda \cdot P_{M_d}(t) + (1 - \lambda) \cdot P_{M_c}(t)\right]$$

Here, $P_{M_d}(t)$, the probability of term $t$ occurring in document $d$, and $P_{M_c}(t)$, the probability of the term appearing in the whole corpus, are estimated as:

$$P_{M_d}(t) = \frac{\text{tf}(t, d)}{\text{dl}(d)}$$
$$P_{M_c}(t) = \frac{\text{cf}(t)}{T}$$

Where $\text{tf}(t, d)$ is the term frequency of term $t$ in document $d$, $\text{dl}(d)$ is the document length (the number of tokens) in document $d$, $\text{cf}(t)$ is the collection frequency of term $t$, and $T$ is the number of tokens in the whole collection.

The query `"teach teach education campus"` is made to an retrieval system over the following corpus:

**Doc 1**: `"teach education school university education"`
**Doc 2**: `"education education campus teach teach"`
**Doc 3**: `"university road school teach learning"`
**Doc 4**: `"campus learning education learning"`

1. Calculate the probabilities $P_{M_{d_i}}(t_j)$ and $P_{M_c}(t_j)$ for all terms $t_j$ and documents $d_i$.
2. Calculate the ranked result set according to the unsmoothed, uniform model $P_{\text{uni}}(d \mid q)$.
3. Calculate the ranked result set according to the linear-interpolated, uniform model $P_{\text{interp-uni}}(d \mid q)$ with $\lambda = 0.5$.

To ease correction, whenever you are calculating something for all terms, sort terms alphabetically.

[1] Linear interpolation is also known as Jelinek-Mercer smoothing.

## 2 n-gram Models (10 Points)

In Task 1 we have estimated probabilities of a unigram language model $P_{\text{uni}}(t_n)$ as:

$$P_{\text{uni}}(t_n) = \frac{\text{tf}(t_n, d)}{\text{dl}(d)}$$

Unigram language models are characterized in that they treat each occurence of a word in a document as independent of all other words. That is, in a unigram model we can calculate the probability of a sequence of words $t_1 t_2 t_3 t_4 t_5$ appearing in a document as:

$$P_{\text{uni}}(t_1 t_2 t_3 t_4 t_5) = P_{\text{uni}}(t_1) \cdot P_{\text{uni}}(t_2) \cdot P_{\text{uni}}(t_3) \cdot P_{\text{uni}}(t_4) \cdot P_{\text{uni}}(t_5)$$

Obviously, in practice words do not occur independent of each other. For instance, it is much more likely that the word *"retrieval"* follows the word *"information"* instead of the word *"hamburger"*.

We can build better models to facilitate this fact. For example, bigram models, where the probability of a word $t_n$ occuring is conditioned on the previous word $t_{n-1}$, and trigram models, where that probability is conditioned on the two previous words $t_{n-2} t_{n-1}$. Bigram and trigram probabilities can be estimated as:

$$P_{\text{bi}}(t_n \mid t_{n-1}) = \frac{\text{tf}(t_{n-1} t_n, d)}{\text{tf}(t_{n-1}, d)}$$
$$P_{\text{tri}}(t_n \mid t_{n-2} t_{n-1}) = \frac{\text{tf}(t_{n-2} t_{n-1} t_n, d)}{\text{tf}(t_{n-2} t_{n-1}, d)}$$

Where $\text{tf}(t_1 t_2 \ldots t_n, d)$ is the frequency of the term sequence $t_1 t_2 \ldots t_n$ appearing in document $d$.

1. How could we estimate the probability of a term sequence $t_1 t_2 t_3 t_4 t_5$ appearing in a document in bigram and trigram models? In your solution give a general formula on what to calculate and detail in which ways we have to modify the definitions of the model (if applicable).

2. Using your above answer, calculate the probability of the term sequence `"rose is a rose"` appearing in any of the following documents according to a unigram, a bigram, and a trigram model:

   **Doc 1**: `"rose is a rose is a rose is a rose"`
   **Doc 2**: `"rose rose rose rose is is is a a a"`
   **Doc 3**: `"rose is a rose"`
   **Doc 4**: `"a rose is a"`

## 3 Programming (16 Points)

Similiar to Boolean and TF-IDF retrieval, the query likelihood model can also be implemented on top of an inverted index.

However, one adjustment has to be made. If using a smoothed language model, there always is a non-zero probability of each term occuring in every document (whether that term occurs in the document or not). This would require iterating over all documents in the corpus whether they contain a query term or not. We thus make the following change to the query likelihood model: we define that only documents that contain atleast one query term should be returned (all other documents have the same, smaller probability anyways and would just be ranked below the documents that contain atleast one query term).

Your task is to create a new file `query_likelihood_inverted_index.py` and implement the query likelihood model with the linear-interpolated unigram model from Task 1 and $\lambda = 0.5$ over an inverted index (similar to the implementations of the last assignments) in a class `QueryLikelihoodInvertedIndex`. Your class should support the familiar methods `add_document(self, docid, terms)` and `search(self, terms)` where the `terms` parameter should accept lists of word-strings in both cases. The `search()` method should return a list of pairs of document-IDs and the probability of that document being relevant to the query, where that list is sorted after probabilities descending.

You can test your solution using file `test_query_likelihood_inverted_index.py`.

It is recommended to orient your solution around the refernce solutions of the last assignments.

# Important Notes

Ask questions and discussion with regard to the lecture, tutorial, or assignments in

- The WebScience newsgroup:
  https://webnews.uni-koblenz.de/newsgroups.php?group=infko.webscience
- Our Facebook group:
  https://facebook.com/groups/InformationRetrievalUniKoblenz

## Submission

- Solutions have to be checked into the SVN repository. Use the directory name `solutions/assignment5/` in your group's repository *mandatory*.
- The SVN repository is available via
  https://svn.uni-koblenz.de/westteaching/ir-ss17/<groupname>
- Solution format: all solutions for theoretical taks as *one* PDF document. Programming code has to be submitted as Python code.
- The name of the group and the names of all participating students must be listed on each submission:
  - at the top of each PDF file
  - at the top of each Python file (in comments)

  Only named students will receive credit.
- With the submission of your solution you confirm that you created the solution independently as a group, especially without using other intellectual contributions. That is, you submission should not be plagiarism!

## Programming Assignments

The programming assignments require you to have a Python 3.6+ interpreter (older versions may still work coincidentally, but are not officially supported) and the SciPy stack installed. For an installation guide, see http://west.uni-koblenz.de/en/studying/installing-python

The following rules apply for the *implementation* of your solution:

- You can create as many additional code files as you need. You can create as many additional classes or methods as you need (even in the provided code files). However, do not forget to submit *all* `.py` code files to the SVN.
- Check that your code compiles and runs without errors.

- Use `UTF-8` as the file encoding. *Other encodings will not be taken into account!*
- Test your implementation with the provided test cases.

  Passing of all tests is a *necessity* to receive full score, no *sufficiency*. In general, programming against the provided test cases should assist you in finding correct solutions.
- Make sure your code is formatted to be easy to read.
  - Make sure you code has consistent indentation.
  - Make sure you comment and document your code adequately in English.
  - Choose consistent and intuitive names for your identifiers.
- Do *not* use any accents, spaces or special characters in your filenames.