SNLP 2016
Exercise 10

Submission date: 08.07.2016, 23:59

You can use NLTK for text normalization (incl. punctuation removal, stemming/lemmatizatio, stopword removal and tokenization) you need in this exercise sheet. As for lemmatization, we suggest you to use *Pattern*[1] and specially not to use *WordNetLemmatizer* from *nltk* as it performs really bad!

Information Retrieval

In this assignment you will experiment two techniques for information retrieval: Vector Space Model and Language Models.

The dataset

Your dataset is the Cranfield collection. It contains around 1400 abstracts of journal articles (in aerodynamics) and 255 queries. The file cran.all.1400 contains all the abstracts and the file cran.qry contains all the queries. You will have to parse these files and use the .W part of each abstract and query. The file cranqrel contains the relevant document for every query. The first column is the ID of the query and the second column is the ID of the document. Ignore the third column.

Vector Space Model

The core of every IR system is the inverted index. The inverted index is a dictionary that maps tokens to special structures that contain crucial information about the IR task. The simplest inverted index that you can build, has for every token one list that contains the document IDs that contain that term, and the number of times that this document contains that term e.g.:

$$\boxed{token \rightarrow \Big[\big[docID, \#occurences\big], \big[docID, \#occurences\big], ...\Big]}$$

- (2 points) Implement a function with the name *createInvertedIndex*. This function must take as arguments a file, and should do the pre-processing for the words:
  - Remove stopwords
  - Remove punctuations
  - Lowercase
  - Lemmatization + Stemming

  and make the inverted index of each token in that file.

- (2 points) Now you need to build the document-term matrix. Implement a function *createDocWordMatrix* that takes as arguments an inverted index and an argument for choosing to build a document-word matrix with **term** frequencies **or** with **tf-idf** scores (with which you got familiarized in Exercise 7). You may add additional arguments if you want. After this you should have a matrix whose i-th row and j-th column contains the term frequency or tf-idf score of the j-th word in the i-th document. Notice that you need to have some kind of IDs for the tokens in the invertedIndex, so that you know which word corresponds to which column in the matrix. Documents 994 and 471 are empty. You are allowed to add some irrelevant text to them so that your parsing task is easier.

[1] `http://www.clips.ua.ac.be/pattern`

- (2 points) Perform the IR challenge. Implement a script that computes the similarity between a query and the a document. For similarity measure, use the simplified version of the ***cosine*** similarity:

$$sim(document, query) = \sum_{\substack{\text{term in query} \\ \textbf{or}}} tfidf_{document,term}$$

  For every query you must pick the top-k documents, where k is the number of relevant documents for that query provided in *cranqrel* file (e.g. for the first query there are 29 relevant documents). Consider these k documents as you retrieved document and see how many of them is relevant by *cranqrel* file. Implement a script that reports the **Average Precision** for every query(use definitions on slides 5 in chapter 10...N is our k in here).
  Why do you think we did not use **tf** scores instead of **tf-idf**?

Language Models

In this model, ranking is done by computing the query likelihoods defined as follows:

$$p(q|d) = \prod_{\substack{i=1 \\ \forall t_i \in q}}^{n} p(t_i|d),$$

For this exercise we assume a unigram language model with the following distribution:

$$p(t|d) = \frac{tf_{t,d}}{N}$$

in which N is the size of the document d.
Since you might have terms in your queries that have not appeared in your document ($p(t|d) = 0$), you have to do *smoothing*. Here it is better to do the smoothing with respect to the whole corpus[2], like in *Jelinek-Mercer smoothing*:

$$p(t|d) = (1 - \lambda) \times p(t|d) + \lambda \times p(t|C),$$

Still you might face the cases that your query terms do not appear in the whole corpus, resulting in $p(t|C) = 0$. We can use *Lidstone smoothing* to solve this.

- Use your *createDocWordMatrix* to have **tf** values.
- (2 points) Compute the query likelihoods with smoothing:
  - Jelinek-Mercer smoothing: $\lambda = 0.5$,
  - Lidstone smoothing: $\alpha = 0.1$
- (2 points) Like previous section, keep the top-k most likely documents (top-k $p(q|d)$s) as retrieved and report the uninterpolated **Mean Average Precision** (MAP) (use definitions on slides 6 in chapter 10) this time.

.

Bonus

(1 point) What does precision and recall represent? What does a typical precision vs recall plot look like? Why is there such a relation?

[2] corpus = all documents

1 Submission Instructions: Read carefully

- You can form groups of maximum 3 people.
- Submit only 1 archive file in the ZIP format with name containing the MN of all the team members, e.g.:

Exercise_01_MatriculationNumber1_MatriculationNumber2_MatriculationNumber3.zip

- Provide in the archive:
  - your code, accompanied with sufficient comments,
  - a PDF report with answers, solutions, plots and brief instructions on executing your code,
  - a README file with the group member names, matriculation numbers and emails,
  - Data necessary to reproduce your results [3]
- The subject of your submission mail must contain the string [SNLP] (including the braces) and explicitly denoting that it is an exercise submission, e.g:

[SNLP] Exercise Submission 08

- Depending on your tutorial group, send your assignment to the corresponding tutor:
  - Sedigheh Eslami: *eslami@mpi-inf.mpg.de*
  - Naszdi Kata: *b.naszadi@gmail.com*
  - Stephanie Lund: *stflund@gmail.com*

2 General Information

- In your mails to us regarding the tutorial please add the tag [SNLP] in the subject accompanied by an appropriate subject briefly describing the contents.
- Feel free to use any programming language of your liking. However we strongly advise in favor of Python, due to the abundance of available tools (also note that Python3 comes with an excellent native support of UTF8 strings).
- Avoid using libraries that solve what we ask you to do (unless otherwise noted).
- Avoid building complex systems. The exercises are simple enough.
- Do not include any executable files in your submission, as this may cause the e-mail server to reject it.
- **In case of copying, all the participants (including the original solution) will get 0 points for the whole assignment. Note: it is rather easy to identify a copied solution. Plagiarism is also not tolerated.**
- **Missing the deadline even for a few minutes, will result in 50% point reduction. Submission past the next tutorial, is not corrected, as the solutions will already be discussed.**
- **Please submit in your solutions necessary to support your claims. Failure to do so, might results in reduction of points in the relevant questions.**

[3] If you feel that these files are beyond reasonable size for an email submission and also reasonably convenient, please provide a means for us to access them online

- Each assignment has 10 points and perhaps some bonus points (usually 2 or 3). In order to qualify for the exams, you need to have 2/3 of the total points. For example, in case there are 12 assignments, you need to collect at least 80 out of the 120 points to be eligible for the exams. A person that gets 10 plus 2 bonus points in every exercise, needs to deliver only 7 assignments in order to be eligible for the exams, since 7*12=84.
- Attending the tutorial gives 2 points increase for the corresponding assignment.
- Exercise points (including any bonuses) guarantee only the admittance to the exam, however have no further effect on the final exam grade.