Web Information Retrieval

Assignment 4

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: May 30, 2017, 10:00 a.m.
Tutorial on: May 30 and June 2, 2017

This assignment covers the *Vector Space Model* in Information Retrieval. As supplementary material to the lecture, I can recommend reading *Chapter 6* 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 TF-IDF Calculation (10 Points)

Given the following corpus:

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

Calculate a vector space model over tf-idf weights for this corpus. In particular, calculate the following:

1. The *document frequency* df_t of all terms, i.e. in how many documents each term occurs.
2. The *inverse document frequency* defined as $\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$, where df_t is the document frequency of that term and N is the corpus size, i.e. the number of documents in the corpus.
3. The *term frequency* per document, i.e. which terms occur how many times in each document.
4. The *tf-idf vectors* for each document.

In your answers for all subtasks, sort terms alphabetically.

Now, the retrieval system is faced with the query `"teach teach education campus"`.

1. Calculate the *tf-idf* vector of that query.
2. Calculate the *cosine similarities* between the query vector and all document vectors.
3. Specify the ranked document set that would be the result of the query.

2 Relevance Feedback (6 Points)

Consider the Rocchio method for relevance feedback as discussed in the lecture.

1. In Rocchio's algorithm, what weight setting for $\alpha/\beta/\gamma$ does a *"Find pages like this one"*-search correspond to?
2. Give three reasons why relevance feedback has been little used in web search.

3 Implementating TF-IDF (24 Points)

In this task you are given a partial implementation of an inverted index of a tf-idf vector space model in file `tf_idf_inverted_index.py`. The given code implements only the `add_document()` method, which indexes term and document frequencies for given documents.

3.1 Calculate Document Lengths (7 Points)

Implement the function `calc_document_lengths()`. The function should populate the dictionary `self.document_lengths`, which should map from a document ID to the euclidean length of its vector representation. The length of a document is defined (as in the lecture) as:

$$|\vec{d}| = \sqrt{\sum_t \left(\mathrm{tf}_{t,d} \cdot \log_{10} \frac{N}{\mathrm{df}_t}\right)^2}$$

For your solution, you may only read from the class attributes `self.term_frequencies`, `self.document_frequencies`, and `self.corpus_size`, that were populated in method `add_documents()`. More importantly, you may not modify that method!

For this task you may not `import` any non self-written modules, except the imports already present in the file, i.e. `log10` and `sqrt`.

You can test your solution using the tests provided in class `TestCalcDocumentLengths` in file `test_tf_idf_inverted_index.py`.

3.2 Implement Score Calculation (7 Points)

Implement the function `search()`. This function takes a query (as a list of its words) and should return pairs of matching documents together with their relevance score, sorted descendingly after the relevance score.

The sorting logic is already implemented. For your solution, you will just have to populate the dictionary `scores`, which should map document IDs to the relevance score for that document.

The scores should be the cosine similarities between the vector representations of the document and the query (as discussed in the lecture). As a tip, the lecture slides specifically detail how scores are to be computed in an inverted index implementation. However, make sure to also include the (not ranking influencing) query length factor in your implementation, in order to get correct tf-idf scores.

For your solution, you may only read from the class attributes `self.term_frequencies`, `self.document_frequencies`, and `self.corpus_size`, that were populated in method `add_documents()`. More importantly, you may not modify that method! Of course you may also use the values from `self.document_lengths` which you computed in task 3.1.

For this task you may not `import` any non self-written modules, except the imports already present in the file, i.e. `log10` and `sqrt`.

You can test your solution using the tests that were provided in class `TestSearch` in file `test_tf_idf_inverted_index.py`.

3.3 Implement TF-IDF Variants (10 Points)

There are many variants of how to implement tf-idf vector space models. The SMART notation is a system for naming some popular ones. It takes the form *ddd.qqq*, where the first triplet gives the term weighting of the document vector, while the second triplet gives the weighting in the query vector. The first letter in each triplet specifies the term frequency component of the weighting, the second the document frequency component, and the third the form of normalization used. See the following table for some choices for each component[1]:

Term Frequency	Document Frequency	Normalization
n (natural): $\mathrm{tf}_{t,d}$	n (no): 1	n (none): 1
l (logarithm): $1 + \log(\mathrm{tf}_{t,d})$	t (idf): $\log \frac{N}{\mathrm{df}_t}$	c (cosine): $\frac{1}{\sqrt{w_1^2 + \ldots + w_M^2}}$
a (augmented): $0.5 + \frac{0.5 \cdot \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf): $\max(0, \log \frac{N - \mathrm{df}_t}{\mathrm{df}_t})$	
b (boolean): $\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$		

The lecture presented the most popular variant *ntc.ntc* (which was also the one you should implement in the previous tasks).

For this task, for each of the variants below, copy the file `tf_idf_inverted_index.py` into a new file `tf_idf_inverted_index_<variant>.py`[2] and modify its contents to implement that variant vector space model (keep the class name `TfIdfInvertedIndex`).

- *nnn.atc*
- *ntc.atc*
- A variant of your choice that is not one of the already implemented ones.

For this task you may not `import` any non self-written modules, except the imports already present in the file, i.e. `log10` and `sqrt`.

[1]For more information, see *Section 6.4* of “Introduction to Information Retrieval” by Manning, C. D., Raghavan, P., and Schütze, H. (2008).

[2]Replace the dot in the variant name with an underscore, i.e. save varient *nnn.atc* in `tf_idf_inverted_index_nnn_atc.py`.

Afterwards, you can use file `variant_evaluation.py` to calculate the mean precision@20 and mean average precision scores for your implementations. It will automatically detect your implementations and score them (if you followed the naming conventions). It uses the test collection LISA: a corpus of titles and abstracts of 6000 scientific papers along with 35 queries together with lists of relevant documents. In order to run the evaluation, download that corpus from `http://ir.dcs.gla.ac.uk/resources/test_collections/lisa/` and extract it into a subfolder `lisa`. Copy the output of the evaluation program into a file `variant_evaluation.txt` and commit it together with your solution.

Do not commit the downloaded LISA corpus to the SVN!

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/assignment4/` 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.