Web Information Retrieval

Assignment 1

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 9, 2017, 10:00 a.m.
Tutorial on: May 9 and May 12, 2017

1 Machine Learning and its relation to Information Retrieval (14 Points)

1. Research on your own and give a definition on what it means for a computer porgram to *learn* in the context of Machine Learning.
2. Explain the difference between *supervised* and *unsupervised* learning.
3. Explan the difference between *flat* and *hierarchical* clustering.
4. Research or come up with a scenario in which Machine Learning techniques could be used in Information Retrieval and describe it.

Note: In this course, the task with such general theoretic questions is for you to describe just the idea that is central to a certain concept—not every intricate detail (we are not expecting research papers as answers). This task can probably be answered with 1-2 paragraph per subtask. This is also true for the theoretic tasks in all following assignments (unless otherwise noted).

2 Naive Document Search (10 Points)

Your task is to implement the naive document search algorithm for finding documents which contain a given query keyword.

The naive search algorithm works as follows:

1. Iterate over all documents in a corpus.
2. Iterate over all words in the current document.
3. If the current word is identical to the given query keyword (ignoring differences in case) the document matches the search query.

For your solution, implement the function `naive_search(filename, keyword)` inside `naive_search.py`. This function accepts the parameters `filename`, which is a path pointing to a valid document corpus file, and `keyword`, which is a single word that documents should be searched for. The function should return a Python set of *title*s of documents that contained the keyword in their *content* field. You can test your solution using the provided tests in `test_naive_search.py`.

A valid document corpus file is one that follows the following custom XML dialect:

```
<corpus>
<doc id="{doc-id-here}" url="{doc-url-here}" title="{doc-title-here}">
{doc-content-here}
</doc>
<doc id="{doc-id-here}" url="{doc-url-here}" title="{doc-title-here}">
{doc-content-here}
</doc>
...
</corpus>
```

In this course we will be using the Simple English Wikipedia[1] to have a somewhat *real-life* example of a corpus. You can download some corpus files (which are also used by the provided test cases) generated from the Simple English Wikipedia in the correct format from: https://west.uni-koblenz.de/sites/default/files/studying/courses/ss17/WebIR/simplewiki-20160501-extracted.7z

To extract this archive you will need 7-zip. Place these files in the same folder where your Python files are located. When uploading your solution to SVN make sure you are not committing these corpus files! This archive should contain the following files:

- `simplewiki-20160501-extracted.xml`: This file contains the whole Simple English Wikipedia corpus in our corpus format.

[1] https://simple.wikipedia.org/

It was created by first downloading the openly available dataset[2] of the Simple English Wikipedia from May 2016, and second parsing the MediaWiki markup to plain text using the Medialab Wikipedia Extractor[3]. The corpus file format also stems from this tool.

- `simplewiki-20160501-extracted-devel.xml`: This file contains only the first 100 documents from the whole corpus. It can thus be used to quickly test your implementation during development.
- `simplewiki-20160501-extracted-`i`.xml`: These files contains only the first $10\,000 \cdot i$ documents from the whole corpus. They will be used in Task 3.

But beware, the tool used to construct these XML files actually does not produce valid XML: they contain stray ampersands (`&`) and angle brackets (`<` and `>`). So you should probably not use an XML parser, but instead traverse the files manually line-by-line.

For this assignment, among other things you will have to tokenize content read from the corpus into words. To simplify this assignment, you should tokenize strings strictly by splitting at whitespace, i.e. no separate handling of special characters. For example, the string `"Hello, how are you?"` should be tokenized into `["Hello,", "how", "are", "you?"]`. You can do this in Python with str.split.

You also might want to use a function like str.lower to implement the case-insensitive matching.

[2] https://dumps.wikimedia.org/simplewiki/20160501/
[3] https://github.com/attardi/wikiextractor

3 Runtime Analysis (16 Points)

Your task is analyze to correlation between the runtime of the naive search algorithm (implemented in Task 2) and the filesize of the searched corpus.

Specifically, you should create a new Python programm that

1. Lets the user enter an arbitrary query keyword (e.g., via input()).
2. Iterates over the ten `simplewiki-20160501-extracted-`*`i`*`.xml`-files and
   - Searches the current file for the given query keyword using the `naive-search` function from Task 2.
   - Keeps track of the file's file size in Mebibytes (MiB) and the time it took to execute the search in seconds per corpus.
3. Creates a plot of the recorded pairs of corpus file size and elapsed search time using matplotlib and saves it to a file called `elapsed_time_per_filesize.png`.

For a short tutorial on how to plot using matplotlib, see `https://matplotlib.org/users/pyplot_tutorial.html`

For your solution also commit the created plot graphic to the SVN. Make sure that your graph's axes are labeled!

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/assignment1/` 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.