Homework 4:
Text as Vectors, Unit Tests

Dr. Benjamin Roth
Symbolische Programmiersprache

Due: Tuesday November 28, 2017, 16:00

In this exercise you will:

- Practice the list comprehensions and dictionary comprehensions in Python
- Review how to represent documents as vectors, and compare similarity
- Get some hands-on experience using the python `doctest` and `unittest` frameworks

Exercise 1: List and Dictionary Comprehensions in Python [2.5 points]

In this exercise you will solve 5 Tasks to practice a powerful feature of Python: comprehensions. With these, multiple-line for-loop constructions can be expressed in expressive one-liners.

Solve the following tasks given in `comprehensions.py`. You can test the funcitonality of your code calling (from your ./src directory):
`python3 -m unittest -v hw04_text_search/test_comprehensions.py`

1. Increase each value in orig by 2. [0.5 points]
2. Create a list that contains the squares of all uneven numbers in orig. [0.5 points]
3. Create a list that contains the squares of the uneven numbers and the fivefold (das Fünffache) of the even numbers in orig. [0.5 points]
4. Create a mapping from the cubes of the values in orig to the values themselves. [0.5 points]
5. Create a mapping from all elements of wordlist that start with an uppercase letter to their respective length. [0.5 points]

Exercise 2: Search Engine: Running the code

In the source folder for this exercise (`src/hw04_text_search/`), you will find the classes to represent documents, and a simple search engine, which were discussed in the lecture (`text_vectors.py`). There is also a script to interactively search all `*.txt` files in a directory (`interactive_search.py`). Try to understand what each of the classes are doing.

On the course homepage, you can find a dataset of corporate emails[1], containing several folders of spam or normal (ham) emails. Download and unpack it into the `src/data/` folder of your project. Run the interactive search on a email folder (always call scripts from the `src/` folder):

```
python3 -m hw04_text_search.interactive_search --dir data/enron/enron1/ham/
```

Exercise 3: Doctest and documentation

Exercise 3.1: Doctest [2 points]

Use the doctest module to write tests for the functions `dot` and `normalized_tokens` in the module `hw04_text_search.text_vectors`.

Run your tests with:

```
python3 -m doctest -v hw04_text_search/text_vectors.py
```

Exercise 3.2: Docstrings [5.5 points]

Provide docstring documentation for all member functions (including constructors) of the classes `TextDocument`, `DocumentCollection` and `SearchEngine` in the same module.

Exercise 4: Extending the program using test-driven development [9 points]

Improve the program by adding additional functionality. Use the `unittest` framework, and extend the module `hw04_text_search.test_text_search`. You should add tests that initially fail, and only pass once you successfully added the missing functionality.

Have a look at the example for a test given with:
`DocumentCollectionTest.test_unknown_word_cosine`
This test fails, as you can verify by running:

```
python3 -m unittest -v hw04_text_search/test_text_search.py
```

- Make the existing test pass by changing the functionality of `DocumentCollection.cosine_similarity` accordingly. **[1 point]**
- Write 4 additional tests that initially fail, and then pass after some functionality of (any part of) the initial code has been changed/extended. **In order to get full credits, your test must fail on the initial code, and pass on the changed**

[1] See https://en.wikipedia.org/wiki/Enron_Corpus for the history of this dataset

code that you check in. The test must also contain a short docstring describing what is being tested. [8 points]

You can come up with your own improvements to the code, or you can choose from the following list (in each case also write the appropriate test):

- The search engine displays text snippets including line break. Change the functionality such that lines are displayed without line breaks.
- Remove the indentation markers of reply emails (e.g. "`> > > > `") (either when reading or when displaying).
- If several search terms occur in a document, the search engine displays several text snippets (one for each). Change the code such that only one text snippet is displayed, if it contains the entire search string.
- Query syntax: if tokens are quoted (`"New York"`), require that full string occurs in the document (hint: additionally filter result of `docs_with_tokens`)
- Snippets should show exact matches of query tokens, not substring matches.
- Files to index should recursively be read from subdirectories.
- When the file path is shown for search result, normalize it so that the full absolute path is shown.
- If there is no result containing all tokens, search for documents containing at least one of the tokens.
- If a query contains the same token multiple times, only show one text snippet for it.