Homework 8: Comprehensions, Text as Vectors, Tests

Robert Litschko[*]
Symbolische Programmiersprache

Due : Thursday, December 15, 2022, 12:00 (noon)

In this exercise you will:

- Practice list and dictionary comprehensions in Python
- Review how to represent documents as vectors, and compare similarities
- Write your own python `doctest` and `unittest` tests

Exercise 1: Comprehensions in Python (recap) [5 points]

In this exercise you will solve 5 Tasks to practice Python comprehensions[1]. With these, multiple-line for-loop constructions can be expressed in expressive one-liners. Solve the following tasks given in `comprehensions.py` using list or dictionary comprehensions.

You can test the functionality of your code calling:

```
python3 -m unittest -v hw08_text_search/test_comprehensions.py
```

1. Complete the function `multiply_by(x, list1)` that multiplies each value in a list by x and returns it as a new list. [1 point]
2. Complete the function `check_division(x, list1)` that takes a list and returns a list indicating whether or not each element in the original list can exactly be divided (without remainder) by x (e.g check_division(3, [1,2,3]) -> [False, False, True]) [1 point]
3. Complete the function `div_less(set1)`. It should return a new set containing only those numbers from set1 that can't be divided by any other number from set1 except one and itself (again, division without remainder). [1 point]

[*]Credit: Exercises are based on previous iterations from Katerina Kalouli.
[1]`https://www.geeksforgeeks.org/comprehensions-in-python/`

4. Implement the function `map_zip(list1, list2)`. It should return a dictionary mapping the *nth* element in list1 to the *nth* element in list2. Make use of the `zip()` function in your dictionary comprehension, that can handle lists of different sizes automatically. [1 point]

5. Complete the function `word_to_length(list1)`. It returns a dictionary mapping all words of the list with at least 3 characters to their number of characters. [1 point]

Exercise 2: Search Engine: Running the code [2 points]

In the source folder for this exercise (`hw08_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. In the `data/` folder of your project, you can find a dataset of corporate emails (enron)[2], containing several folders of spam or normal ("ham") emails. Run the interactive search on an email folder, e.g., :

```
python3 -m hw08_text_search.interactive_search --dir ../data/enron/enron1/ham/
```

(`--dir` should contain the whole path to your data folder)

Note that if you run this script, you will be prompted to enter a query, i.e., something that should be searched in the emails. If you enter a query and you hit Enter, you will receive an error message because at this point the implementation of the method cosine_similarity() is still missing (we leave this for Exercise 3). For this task, use the doctest module to write at least one meaningful test for each of the functions `dot` and `normalized_tokens` in the module `text_vectors.py`. [2 points]

Exercise 3: Extending the program [4 points]

Improve the program by adding additional functionality listed below. You can test the functionality of your code calling:

```
python3 -m unittest -v hw08_text_search/test_text_search.py
```

Tasks:

1. Make the existing test pass by changing the functionality of `cosine_similarity` inside `DocumentCollection` accordingly. [1 point]

2. Preprocessing: The Search engine currently displays text snippets including line breaks. Change the functionality such that lines are displayed without line breaks (take a look at the unittest to see an example). Implement your changes in the `TextDocument` class. [1 point]

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

3. If there is no result containing all tokens, the search engine should return tokens containing at least one of the tokens. Implement `docs_with_some_tokens` in `DocumentCollection` and update the `ranked_documents` function inside the `SearchEngine` class correspondingly.[3] [1 point]

4. Modify the `snippets` function inside `SearchEngine` to implement the following functionality. If all query tokens (i.e., the query string) can be found in a document, highlight them together instead of individually.[4] For example, let's assume the query is "sat on" and the document is "the cat sat on a mat".

   - Your implementation should return the snippet "... cat [sat on] a ma..." for the document
   - The current implementation returns ["... cat [sat] on a...", "... sat [on] a ma..."].

   [1 Point]

[3] For this unittest to work you need to unpack enron.zip in your data folder

[4] Optional: Familiarize yourself with python generators to understand the meaning of the `yield` keyword inside the `snippets()` function (https://www.programiz.com/python-programming/generator)