Programming

Winter 2020/2021

Number 04, Submission Deadline: Dec. 08, 2020

1. Jupyter Notebook (2P)

Use Jupyter Notebook for preparing your solutions to this exercise sheet. Submit your solutions as both

1. Jupyter Notebook and
2. PDF printout of your Notebook.

2. Processing text from Word (.docx) documents (3P)

Often, text is not stored plainly, but in some formatted structure, such as Word documents, HTML web pages, or XML files. In this exercise, you will learn

- How to install third-party packages with Anaconda (1P)
- How to preprocess such data through the example of reading a Word document. To this end, you are provided with the first chapter of Mary Shelley's novel *Frankenstein* in file `Frankenstein.docx`. (2P)

Note that a *.docx* file is a compressed archive that contains a set of different XML documents that contain content, but also theme details, settings and style instructions:

```
Browsing zipfile Frankenstein.docx:
[Content_Types].xml
_rels/.rels
word/_rels/document.xml.rels
word/document.xml
word/theme/theme1.xml
word/settings.xml
word/fontTable.xml
word/webSettings.xml
docProps/app.xml
docProps/core.xml
word/styles.xml
```

The third-party package `python-docx` provides a Word Document parser that you will use to load the contents of `Frankenstein.docx` into your Python program.

1. Open Anaconda Navigator and switch to the "Environments" view.
2. Add *channel* "conda-forge" by clicking on the "channels" button, and then "add".
3. Update the (package) index afterwards.
4. Switch to the "Not installed" list and search for "python-docx", then select the package for installation and hit the "Apply" button.

With the package being installed, turn to the task of processing the `Frankenstein.docx` file:

1. Read up on the package and its API (Application Programming Interface) at https://python-docx.readthedocs.io/.
2. Load the `Frankenstein.docx` file.
3. Iterate over its `paragraphs`, and extract the content in a single, long string. Delimit paragraphs by a newline (`\n`) character.
4. Print the contents of the string.

3. Improving the lemmatization of the Frankenstein text (10P)

In the lecture, I have noted that lemmatization without Part-Of-Speech (POS) information will lead to a poor performance. You will now investigate this issue further in this exercise.

- Repeat the analysis of the lecture in which the WordNetLemmatizer is applied to the Frankenstein text. (1P)

The process of obtaining POS information is called `tagging`. Further information on tagging with NLTK is provided at https://www.nltk.org/book/ch05.html.

This exercise requires additional data from NLTK's database, which can be obtained by executing the following Python commands:

```
import nltk
nltk.download('averaged_perceptron_tagger')
nltk.download('universal_tagset')
```

Use NLTK's functionality to tag the Frankstein text. Proceed as follows:

1. Tagging can only be performed sentence-by-sentence. Therefore, alter the analysis shown in the lecture to create a list of sentences, where each sentence is again represented by a list of its contained words that preserves their original order. Use NLTK's sentence and word tokenizers (as shown in the lecture) for this task. (2P)
2. Tag each sentence using NLTK's `pos_tag()` function. When calling this function, use the parameter setting `tagset='universal'` to indicate the use of the more commonly used *universal* POS annotation (called *tagset*) instead of NLTK's default one. (1P)
3. Before you can supply the now obtained POS information to the WordNetLemmatizer, the tagset must be mapped to WordNetLemmatizer's tagset dialect. The WordNetLemmatizer can only lemmatize *nouns, verbs, adverbs,* and *adjectives*. Create a dictionary to map between the universal tagset and WordNetLemmatizer's tagset dialect using the following table (1P):

NOUN	n
VERB	v
ADV	r
ADJ	a

4. Create a function `lemmatizeSentences(text, universal_to_wordnet_tagset)` that lemmatizes the given text (represented as lists of word-tokenized sentences) and your previously created mapping between the universal and WordNetLemmatizer's tagset. Discard any word whose tag does not appear in the mapping. (2P)
5. Print the 10 most frequent lemmatized words after tagging. (1P)
6. Compare the *sets* of lemmatized words (i) before and (ii) after tagging. (2P)