Programming Assignment 2 – Basic Probability, Computing and Statistics 2016

Fall 2016, Master of Logic, University of Amsterdam

Instructors: Philip Schulz, Christian Schaffner and Bas Cornellisen

Submission deadline: Wednesday, November 16th, 8 p.m.

Note: if the assignment is unclear to you or if you get stuck, do not hesitate to contact Philip.

1 Exercise

This week we will start to work with some real data. The website Project Gutenberg provides access to thousands of books which have run out of copyright. They are freely downloadable for you to read. Our goal is to analyse these books for some of their basic properties. In particular, we want to find their most frequent and least frequent words and also find the counts for specific words. For example, we might want to know how often the word *Alice* occurs in Alice in Wonderland.

Just as last week we will again write an interactive program. This time, our program will be able to read text files from the command line. Moreover, it will keep interacting with the user until the user explicitly tells it to stop. During development you can experiment with any book you like. However, when grading we will all use this version of War and Peace.

As you will notice, the file contains some document information from Project Gutenberg. This means that our word counts will not be entirely correct as there are also words that do not belong to the main text. We will not be bothered by this, however.

One last remark: when you download any book of your choice, please make sure to download the `Plain Text UTF-8` version of it. Otherwise your program will not be able to read the file (at least not without some additional effort).

Here is what your program must be able to do:

- Ask the user for a path to a text file and read that text file into memory
- Ask the user what he wants to do. The user should be able to choose an operation by typing the appropriate number:

1. Look for the most frequent words
2. Look for the least frequent words
3. Look for the count of a specific word
4. Exit

- After each operation has been finished, the program should show the user the initial choice again. The user should then be able to perform further operations until he decides to exit. Notice that reading the text anew for every operation would be extremely wasteful. You should try to get your program to memorize all relevant information.

2 Grading

The task is to count words regardless of casing. This means that *The* and *the* should be treated as the same word. To achieve this, all words should be lowercased before counting them. However, we are not going to worry about punctuation. For us *Alice* and *Alice*? are different words. We define a word as any sequence of characters that is divided from other words by white spaces.

1 point When the program starts it asks the user for a path to a file (including the file name!). It is ok for the program to crash if this path does not exist.

2 points The program reads the text file only once and stores all relevant information before showing the user any options. This has to be checked analytically by inspecting the code. Depending on your computer reading a file can take several seconds.

1 point Whenever the user enters an invalid value, the program responds "Sorry, I do not understand this. Please choose again." and lets the user re-enter the value.

2 points When the user asks for a list of most frequent words, the program responds by asking for the number n of most frequent words to produce. It then outputs the n most frequent words to the screen as a list in which each line looks as follows:

$$word \quad wordCount$$

If there are several words of the same count, they are ordered alphabetically (1 point). For better readability you should insert a tab in between the word and its count (this is not relevant for grading, though. Any whitespace will do fine).

2 points When the user asks for a list of least frequent words, the program responds by asking for the number n of least frequent words to produce. It then outputs the n least frequent words to the screen as a list in which each line looks as follows:

$$word \quad wordCount$$

If there are several words of the same count, they are ordered alphabetically (1 point). For better readability you should insert a tab in between the word and its count (this is not relevant for grading, though. Any whitespace will do fine).

2 points The count of specific words is printed in the same format as the above two items (1 point). If a word does not occur in the text, the program prints *Sorry, this word does not occur in the text* (1 point).

Notice: for the counts, lists of the 1000 most frequent and 1000 least frequent words will be provided to you during the review period. We will also give you detailed instructions on how to compare two lists. To test the specific word functionality, you can just pick a couple of words from these lists.

3 Counters

For this exercise, it may be helpful to use a Python data structure known as Counter. It is not strictly necessary to use counters for this exercise but they might make your life a lot easier. Also make sure to the check the official documentation.