Practical Assignment PR02
Document Handling, Stop Words, Basic Boolean Retrieval

Instructions

1. This assignment is a programming assignment. Your task is to implement all features described in the sections *Task 1–4* below.
2. Download the project files (ir25_pr02_template.zip) from the Moodle page and review them carefully. These files are essential for the assignment, as they include unit tests that will be used to verify the correctness of your implementation.
3. Your implementation must meet a set of requirements that are listed below. Otherwise, it will *automatically* be scored as unsatisfactory, so please pay special attention to this:
   a) Your code must be compatible with our unit tests. To achieve this, modify the file test_wrapper.py so that it correctly connects your internal implementation to the interface expected by the tests! You are free to organize your code in any structure you like. However, the functions defined in test_wrapper.py must correctly call your own code. Do not modify any of the test files in the public_tests/ folder!
   b) The program has to run without crashing and without entering infinite loops.
   c) We require your submission to be runnable using Python 3.10.
   d) Use meaningful code comments and identifier names!
   e) Avoid using absolute paths or platform-specific path syntax (e.g. slashes/backslashes) in the source code! The program should be executable without problems and completely even after changing the computer or operating system!
4. Submit your source code documents as a ZIP archive via Moodle. You will find the deadline for the submission there. Please also ensure the following requirements for your submission:
   a) Name the file as follows: group_*n*_p*m*.zip, where *n* is your group number and *m* is the number of the practical task sheet.
   b) Please place your source files directly in the root of the ZIP archive, not inside a subfolder.
   c) If you use a virtual environment, do not include the venv folder in your submission!

At the end of this document you will find a list of frequently asked questions.

Task 1 – Basic IR System

The goal of the remaining task sheets is to implement a rudimentary information retrieval system. Before implementing the individual functionalities, you must set up the basic structure of your information retrieval system. To this end, write a program that sets up the overall structure required to fulfill the requirements listed in Task 2 of Task Sheet 1. The actual functionality will be implemented in later tasks.

Requirements

1. Create a simple (text-based) user interface that allows users to:
   - Download and parse a story collection from a URL (see Task 2)
   - Search for documents using a single Boolean keyword query (see Task 3)
   - Apply stop word removal (see Task 4)
2. Organize your code into clean, reusable functions and classes. The UI code must only handle user interaction - all processing logic should be located in separate modules or classes.
3. You may leave functionality that is not yet covered in this task sheet as stubs to be implemented later.

Task 2 – Preparation of the document collection

Write a function that downloads a plain-text file from a given URL and extracts individual stories or chapters from it! The file will typically contain collections such as fairy tales, fables, or legends (e.g., from Project Gutenberg). Enable the user to call this function through the UI, so that they may build a collection to search in.

Requirements

1. The function will pe parameterized with a URL pointing to a .txt file containing multiple stories, as well as with information on how to split the stories and the author and origin name.
2. The function must then:
   a) Download the file from the URL.
   b) Read the file starting from a specific line and (optionally) ending at a specific line to skip introductory or trailing content.
   c) Split the file into individual stories using a specified separator string.
   d) For each story, identify the title and main text using a defined title-text separator.
   e) Tokenize the story text into terms (basic splitting into words).
   f) Create a Document object for each story.
3. The result should be a set/list/etc. of `Document` objects. Each object should be hold the following data:
   - `document_id (int)`: A unique number that reflects the story's position within the source file , starting with 0.
   - `title (str)`: Title of the story.

- `raw_text (str)`: Full text of the story without line breaks.
- `terms (list)`: A list that contains all terms of the story. Duplicates are included.
- `author`: Author of the story (provided by a function parameter).
- `origin`: Title of the book/source from where the story was extracted from (provided by a function parameter).

Task 3 – Simple Linear Search

Enable your user to search for documents with a simple 1-word query! For this, implement a basic Boolean retrieval model that performs a linear search across a collection of documents!

Requirements

1. Your implementation must support searching for a **single term** using Boolean logic. The model should return all documents that contain the search term.
2. Matching must be case-insensitive.
3. The search is performed using a linear scan through the collection.
4. The function must return a list of tuples, each containing a Document object/ID/etc. and its relevance score (here either 0 or 1).

Task 4 – Stop Word Removal

Make it possible for your user to search for documents with removed stop words! Your solution must support two approaches: one using a predefined stop word list, and one based on J. C. Crouch's frequency-based method. To achieve this, you are required to implement two separate functions, one for each approach.

Requirements

1. All stop word removal must be case-insensitive. You must also remove punctuation and line breaks. Be cautious with apostrophes (e.g., in contractions and possessives).
2. Your list-based function must:
   - Load a stop word list from a specified file.
   - Filter the terms in the given document accordingly.
3. Your frequency-based function must:
   - Compute term frequencies across a provided collection.
   - Select stop words based on frequency percentiles.
   - Filter the terms in the given document accordingly.
4. Do **not** use external libraries like `nltk`. Only use standard Python modules!

Make sure that your solution considers all requirements listed in this file and upload it on Moodle until the specified deadline! Make sure all unit tests run successfully before submitting!

FAQ

Where can I find the resources to help me get started? If you are unfamiliar with Python, you might consider going through some online tutorials first:

- `https://www.python.org/about/gettingstarted/`
- `https://www.w3schools.com/python/python_intro.asp`

All required IR-related knowledge is covered in the lecture slides. For further reading, please refer to the listed literature on the Moodle page.
Note: We do *not* recommend following online tutorials on how to implement an IR system, as there is a chance that you will expose yourself to plagiarism suspicions if your code resembles online code too closely.

Is there a specific IDE or tools that I need to use? No, you may use any IDE you like, or none.

I found an error/bug in the provided code. What should I do? If you encounter mistakes/errors/bugs, do not ignore the affected tasks! Write about the bug in the forum or via email to the course instructor as soon as possible.

Are there speed/efficiency/elegance requirements? The mandatory requirements are listed in this document. Otherwise, your program does not have to be particularly efficient or elegantly coded.

Can I use external libraries or frameworks? In this first assignment, no external libraries are needed to implement the base functionalities, so please refrain from using unnecessary imports. However, you may use GUI libraries if you need them. If you have any other particular need for a certain library, please ask for permission in the forum first.

Can I / Do I have to build a GUI? Implementing a GUI is completely optional. If you choose to do so, ensure a clear separation of concerns: only the GUI classes should handle windows, buttons, and events. All core logic must remain independent and callable without any GUI interaction.

I have existing code from the last time I took the course / from some other occasion. Can I submit that instead? No, the task requirements of previous instances of this class are not completely compatible with the assignments of SS2025.[1]

Before submitting, how can I check myself whether my program works reliably? You can run the provided test cases yourself to test for basic functionality. However, this is not a guarantee for correctness! We recommend that you write your own unit tests to verify your program. However, this is not mandatory. For more informatiom, see: `https://docs.python.org/3/library/unittest.html`

[1]Also, if you passed in the last summer, you do not have to do the assignments again!