Text Library with automatic and semi-autonomous self-organizing system

Beatriz M. Borges R. and Pedro V. Teixeira
MPEI, DETI, Universidade de Aveiro

Introduction

For our final project, we developed a text library with dynamic, automatic and semi-autonomous self-organizing system - on MATLAB.

To **execute** it, run **main.m**, located inside the _*scr* folder. **It's important that the current working directory is the _*scr* folder, not the root of the project, otherwise the program might not run (properly or at all) and that the _scr folder and its subfolders are added to the path.**

To **run the tests**, run **tests.m**, located inside the _*scr* folder. Some more time-consuming tests are commented.

The **implemented modules** and **tests** are listed in the table below.

Hash	cell_reference_hash, reference_hash, enhanced_datahash, file_xor_hashes, xor_hashes, n_independent_hashes
Bloom Filters	VanillaBloomFilter, InsertVanillaBloomFilter, IsMember, CountingBloomFilter, Count, InsertCountingBloomFilter
GUI	FileInfo, ImportFile, Main, ShowLibraryStats
Init	init_bf, init_library, init_ref_count_bf, init_shinglemap
Minhashing	document_min_nth_hash, Minhash, shingle_Minhash
Utils	**file_system_logic:** file_to_shingles, get_dir_files, get_subfolders **library_stats:** get_random_top_word, library_stats **new_book_logic:** add_book_bf, add_new_book, book_is_in_library, estimate_language, estimate_subject, . get_lang_subj_from_lib_book **similarity_between_files:** min_similarity_within_laguage, most_common_words_file, most_similar_books_within_language, most_similar_books, most_similar_reference_books
Control (to run the program)	main, tests
Tests	TEST_bloom_filters, TEST_hash_function, TEST_Minhash_parameters, TEST_underlying_hash_function

We **implemented a system which can automatically detect a text's language, and sort it into one of several categories according to its subject.** It was decided to create a library and if the file does not already belong to the library, it is subsequently added in its corresponding language and subject. To this end *Minhashs*, a vanilla bloom filters (used to check whether a given file already exists in the library), and a counting bloom filter (designed to supplement the choosing of a given text's subject) were used.

The system was implemented and tested MATLAB 8.5 (R2015a) and MATLAB 9.2 (R2017a). For time and impossibility of finding older versions, it was not possible to test the proper operation of the program in older versions.

General View of the System

The system supports **two features: addition of a new text file to the library and statistics retrieval** (on the library's current state). The program is also able to give the user important information about the library.

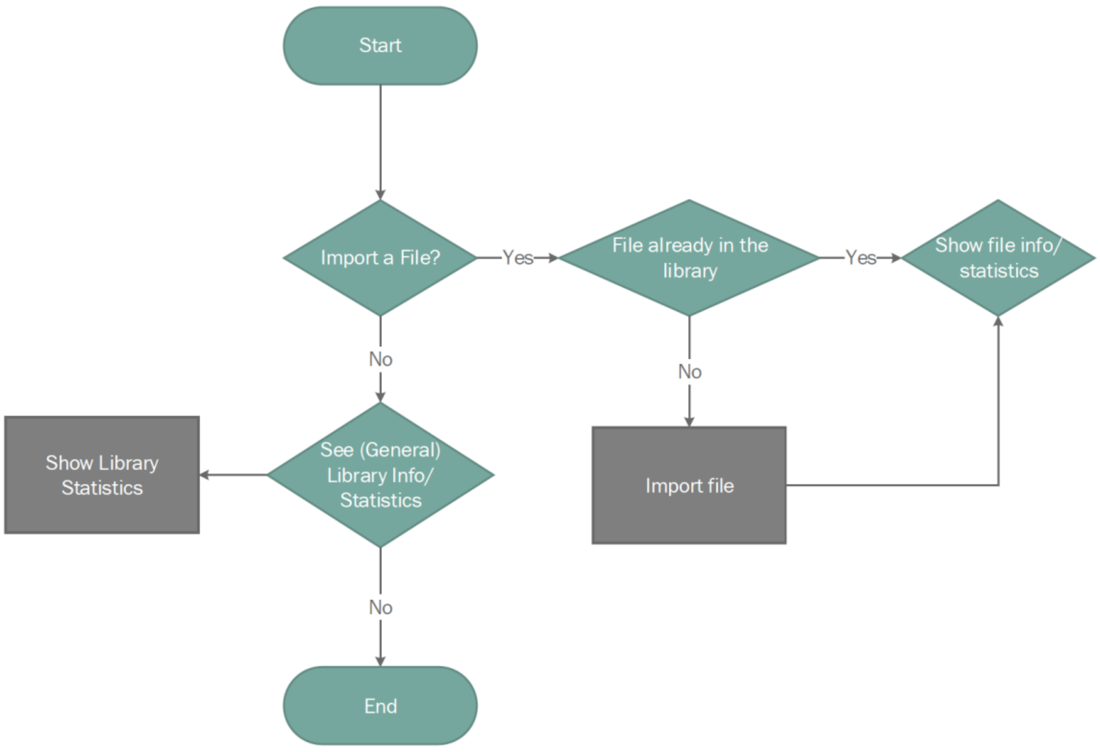
Start
Import a File?
Yes
File already in the library
Yes
Show file info/ statistics
No
No
Show Library Statistics
See (General) Library Info/ Statistics
Import file
No
End

Figure 1: General Flowchart of the Program

Importing a Text File to the Library

Whenever a file is added to the library, it is first checked whether a copy of the file is already present. This is achieved through a (vanilla) bloom filter. If the file is not new to the library, it is first parsed, in order to obtain the necessary information to correctly organize it in the library. Should the file already exist in the library, it is not parsed again.

After importing the file, a window is shown giving the user the **following info about the file**:

1. **Estimated Language** (between Portuguese, English, French, and German);
2. **Estimated Subject** (between Biology, Mathematics and Literature);
3. **Suggestions of another text that might be of interest** (due to their similarity).

Library Statistics

The **library statistics** are a useful collection of general information about the library’s current state, such as:

1. **Number of** text **files** in the library;
2. **Most populated language**;
3. **Most common words** for a random file in each language;
4. **Most populated subjects** for each language.

Detailed View of the System

How a Text File is imported to the Library

The flow of operations the system undertakes when parsing a new file is portrayed by the following diagram:

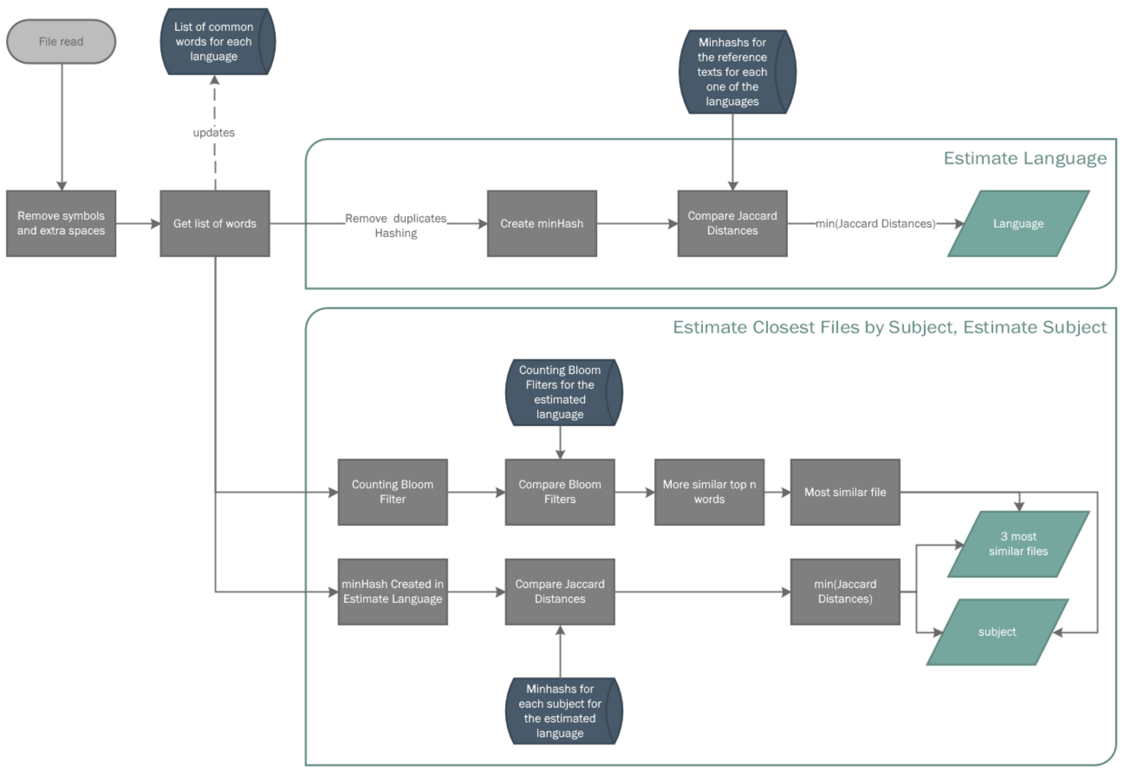
File read
List of common words for each language
updates
Minhashs for the reference texts for each one of the languages
Estimate Language
Remove symbols and extra spaces
Get list of words
Remove duplicates Hashing
Create minHash
Compare Jaccard Distances
min(Jaccard Distances)
Language
Estimate Closest Files by Subject, Estimate Subject
Counting Bloom Filters for the estimated language
Counting Bloom Filter
Compare Bloom Filters
More similar top n words
Most similar file
3 most similar files
minHash Created in Estimate Language
Compare Jaccard Distances
min(Jaccard Distances)
subject
Minhashs for each subject for the estimated language

Figure 2: General Flowchart of the Information Retrieval Processes

Data storage and organization. Creation of the initial Data Set

In Figure 2, two databases are shown:

1. *Minhash* for each language (Portuguese, English, French and German);
2. *Minhash* for each subject (Biology, Mathematics and Literature) for each language.

Both databases were created by processing a representative data set of text files for each one of the languages and its subjects.

The dataset was created using text files freely available on the Internet (see References), that were pre-processed to remove headers, symbols and other characters that would influence the result (such as Greek letters in mathematics texts, that would interfere with the precision of the language estimation since it would be shared symbols between all the languages).

Hash Functions

Hash Functions Possible Choices

There are several possible choices from which to select which hash functions to use, both for the vanilla bloom filter as well as for the *Minhash*. Among them, the following are particularly relevant:

- n independent, uniformly-distributed, autonomously created hash functions;
- one single hash function, whose result is then used for a XOR (exclusive OR) operation with n randomly and autonomously generated numbers;
- a shingle-to-hash reference table, whereby any shingle is only processed once, and after, whenever needed, the hashing results are always looked up in the reference table.

Choosing the variations and the number of Hash Functions

To better decide how many hash functions were chosen for the *Minhashs*, the following (pre) tests' results were taken under consideration:

- The percentage of false positives for whether a given document already exists in the library, noting the number of hash functions that produced the smallest result.
- Temporal performance;

Giving us a optimal value of 50 for use in the *Minhashs.*

For the Bloom Filters, and having in mind the theoretical formula given during classes, we used the given formula defining a collision probability of $p = 1\times 10^{-6}$.

To help determine which hash functions variation to use, the following tests were also taken into account:

- The percentage of false positive for uncorrelated (very dissimilar) languages;
- The percentage of false positive for languages sharing ancestry;
- And the percentage of false positive for similar or very closely related languages.

This revealed the best solution to be the joint use of both *XOR hashes* and *Reference hashes*.

All hash function options were also tested on their independence, distribution and temporal performance.

Post-Implementation Tests

These tests aim to transmit what our project achieved, since it was the feedback from the tests described in the previous subsection that was used to inform the architecture's most important decision (the hash function variation and number). The most relevant indicators to this end were:

- The number of false positives for both language and subject estimation, ~25% and ~45% respectively
- The full setup time: 1 hour and 7 minutes; (~20 minutes for library and *shinglemap* creation, ~47 minutes for the *ref_count_bf* (counting bloom filter) creation, and ~2 seconds for the *lib_bf* (library's (vanilla) bloom filter);
  - These times are only for the first time running the program, if no *.mat* are in the *_scr* directory
- The average time used to process a new file (with 1000 words, on average): 8 minutes, 20 seconds, out of which 4 minutes and 15 seconds were used for finding the *n* most similar files to the new file, out of which the most similar was presented upon conclusion.

Final Remarks

Implemented previously under consideration work

The following topics, previously under consideration, were implemented:

- The joint use of a counting bloom filter and *Minhash* for the best subject estimation results.
- The creation a list of the most common words for each file in the library, by processing the files into a counting bloom filter. Only the 100 most frequent words in the file were considered for its creation. This list was subsequently used in subject estimation.

Future Work

A few features that could expand the library's functionality, but could not be implemented given our development time follow:

1. Add more (supported) languages to the library;
2. Estimate a subject not only by each subject's reference files, but by all the files present in each given subject;
3. Improving efficiency of *most_similar_files_within_language* module by using a combination of counting bloom filters and Minhashing, as opposed to only Minhashing.
4. Possible removal of the most frequent words of each language from a given file before estimating its language and/or subject.

5. Improving the execution time by transcompiling the modules responsible for the hash functions, bloom filters and *Minhash* to a more efficient language, such as C/C++;
6. Usage of the list of most common words for each file in the library both on finding the most similar files to a given reference file, and in language estimation is also possible.
7. Dynamic creation and update of the data set by allowing users, when importing a file, to specify the language and subject of the file (which would improve the system's precision);
8. Allow the user to add new languages;
9. Allow the user to create a new subject;
10. Allow the user to correct a file's classification (that is, allow the user to overwrite a given document's estimated language and subject).

It's important to note that the last four items pose a new possible source of errors, should the user disregard the given instructions or give misleading information.

References

- The Dataset was created using files from Project Gutenberg (https://www.gutenberg.org/catalog/) and LoyalBooks (www.loyalbooks.com).
- The use of online Mathworks documentation and
  Duane C. Hanselman, Bruce R. Littlefield, and Bruce C. Littlefield. 2000. Mastering MATLAB 6: A Comprehensive Tutorial and Reference (1st ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA
  was extremely valuable both to create our functions and the GUI.
- The use of MPEI's presentations for the theoretical values used in the initialization of the bloom filters.
- Matthew Casperson's article on Minhashing (http://matthewcasperson.blogspot.pt/2013/11/Minhash-for-dummies.html) for the inspiration behind the *xor_hashes* module (see 2nd possible choice at Hash Functions Possible Choices).