Programming Project
Document Similarity

Algorithms for Data Science

October 9th, 2020

The objective of this programming assignment is to be able to implement a system for document similarity, using the Min-Hash and Locality Sensitive Hashing methods presented in the lectures.

The programming assignment is *individual*.

1 Preliminaries

The input will be a file which contains a text document on each line:

```
doc1
doc2
...
```

An example of such a file is available at `https://www.lri.fr/~maniu/tweets.txt` and was used during Lab 1. In the following, we assume that documents are 0-indexed, i.e., the first line represents document 0, the second document 1 and so on.

Your task is to implement a command-line program, *in the programming language of your choice*, which takes the following two parameters:

1. the name of the file containing the documents
2. the similarity threshold in $[0, 1]$

The command line syntax is thus the following:

```
./<program_name> <doc_file> <similarity>
```

The console output will be the pairs of document ids that have a *Jaccard similarity* above the threshold, followed by their true similarity number:

```
./<program_name> tweets.txt 0.05
```

will return the following output:

```
0 455 0.062016
...
```

2 Assignments

2.1 Algorithm Implementation

Implement the full workflow for document similarity, as presented in the lecture:

1. transform the documents into sets of k-shingles
2. create the *signature matrix*, using n min-hash function, as exemplified in Lab 1
3. use Locality Sensitive Hashing, by carefully choosing the threshold to minimize the number of *candidate document pairs*
4. compute the *true Jaccard similarity* for the document pairs, and output them if above the threshold

An important note for implementation: *all the algorithms should be implemented by you*, including k-shingling, min-hashing, and LSH.

2.2 Analysis & Experimental Evaluation

For a full submission to the project, you must write a document containing an implementation analysis and an experimental section.

The *implementation* analysis section will contain a write-up of the implementation choices. For example, it should contain an analysis of the data structures used, but also a discussion of the optimizations used in the implementation.

The *experimental evaluation* analysis will contain an empirical evaluation of the algorithm in the form of plots evaluating the execution time, memory used, for a selection of parameters: similairty threshold of LSH, number of documents, document size, etc. Another important parameter to evaluate is to measure the *false positive rate* – the fraction of documents that are chosen for comparison but are not similar). For instance, a possible plot has the number of bands of LSH on the x-axis and the execution time on the y-axis.

The above are just suggestions: other evaluations are welcome and appreciated and will count in the final grade – if they are relevant to the project focus.

3 Submission & Evaluation

Submit you solutions by **Friday, October 30th 2020, 11:59pm** for full credit. Submissions sent by Sunday, November 1st 2020, 11:59pm will incur 5 points of penalty out of 20. Submissions received after this date will receive no credit.

Your submissions should contain two files, and go into two places:

1. an archive (.zip, .tar, .rar, etc.) consisting of the source code and instructions on how to compile (if necessary) and execute the code, which should be submitted in the "Project – Code" homework section
2. a document (.pdf, .doc, etc.) containing the report, which should be submitted in the "Project – Report" homework section

Your submission will be evaluated based on private tests run on your algorithms for correctness, on the evaluation of the code and implementation quality, and the quality of your evaluation document. The implementation and documentation will have equal weight – 50% – in the final grade.