ID2222 Assignment report 1

Authors:

- Marco Dallagiacoma marcoda@kth.se
- Roberto Bampi bampi@kth.se

The purpose of this first assignment is to implement a document similarity system that is able to compute a metric of similarity between different documents and report to the users those that score above a threshold `t`.

The code is implemented as a pure-python module and it is compatible with python versions 3.5 and above. The easiest way to run it is to create a virtualenv first and then install the package into it.

```Bash
# assuming the current directory contains the setup.py file
~ $ virtualenv -p $(which python3) .venv
~ $ source .venv/bin/activate
~ (venv) $ pip install -e .
```

Once the steps above are complete, the package is now installed and ready to run. When run, the program will recursively scan a directory and compare every `.txt` file for similarity (up to a limit of 1000 files by default).

Many parameters can be tweaked, below are some examples of invocations, the complete list of options can be obtained with `--help`:

```Bash
# list options and quit
~ $ check-similarity --help
# check exact similarity with jaccard on 10000 documents
# the check will be executed on the Documents/ directory
~ $ check-similarity --mode exact --file-limit 10000 Documents/
# tweak the number of hash functions for min-hashing and 0.6 similarity
# (default similarity is 0.4)
~ $ check-similarity --mode minhash --threshold 0.6 \
                     --hash-functions 50 Documents
```

Solution

The program is implemented as a Python module and divided into many files. The core algorithms that

perform similarity checking are `minhashing.py`, `exact.py` and `lsh.py`.

Each of the algorithms is implemented as a class with two methods:

```python
def add_document(self, document: Document) -> None
def similar(self, threshold: float) -> Dict[Tuple[str, str], float]
```

`add_document` adds a document to the collection of documents to compare, while `similar` performs similarity checking and returns all the pairs with similarity greater than `threshold` in a dictionary with the form `(doc1, doc2) -> similarity`.

This design enabled all algorithms to share the same interface with enough flexibility to allow *local sensitive hashing* to limit the set of documents to compare.

Algorithms

Exact matching

In exact matching, the document is divided into shingles of three (by default, tunable via parameter) which are then hashed to save space. The hashed shingles are then saved into sets which are then compared using the standard jaccard similarity $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$.

Min-hashing

Min-hashing avoids the expensive set operations by computing a short signature for each set of shingles and comparing the fixed-size signatures instead of the complete set of shingles. The signatures are computed by repeatedly hashing all of the shingles with different hash functions and taking the minimum value for each hash function. The vector of all minimums is the signature.

The hash functions are of the form $(ax + b) \mod c$ where $c = 2^{31} - 1$ (or any other prime number), $0 \leq a \leq c$ and $1 \leq b \leq c$. Therefore, to create any number of hash function, the only requirement is to randomly generate the a and b parameters.

LSH

Both previous algorithm still need to check every document against every other to find matches. Locally sensitive hashing is a probabilistic method to generate a list of candidate pairs that can then be compared for similarity with min-hashing.

To compute the set of candidate documents, the matrix created by using the signatures of all the documents as columns is divided into b bands of r rows (every row is therefore the i^{th} element of each signature). Each column of each band is then hashed toghether and the hash is used to check for collision

in a bucket. If a collision is found, the two documents colliding are candidates and must be checked. This check is repeated with a new bucket for each band.

Sample execution

Below is the execution of the program using all three methods on the folder `Part1/awards_1990/awd_1990_00/` of the dataset provided with the exercise.

```Bash
~ $ check-similarity --threshold 0.80 --method exact Part1/awards_1990/awd_1990
checking similarity with method exact
[0.83] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.83] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.85] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.83] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.83] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.81] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/

~ $ check-similarity --threshold 0.80 --method minhash Part1/awards_1990/awd_19
checking similarity with method minhash
[0.82] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.83] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.83] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/

~ $ check-similarity --threshold 0.80 --method lsh Part1/awards_1990/awd_1990_0
checking similarity with method lsh
[0.83] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.87] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.83] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.85] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.83] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
[0.86] /Users/roberto/projects/kth/data-mining/1/Part1/awards_1990/awd_1990_00/
```