Data Mining, ID2222

Short report on homework 1

Finding Similar Items: Textually Similar Documents

Sanvito Alessandro and Stuart Thuany

November 15, 2021

1 Solution

The language of choice for this homework is Python. Python allows for faster prototyping while providing reasonable performances via built-in functions and vectorization via NumPy.

As suggested in the homework description, we have implemented the three stages of finding textually similar documents based on Jaccard similarity: shingling, min-hashing, and locality-sensitive hashing (LSH), and an additional function to compare the set similarity and signature similarity. In particular, shingling was implemented as a class due to the need of a memory state, and the others were implemented as functions.

The class *Shingling* takes as input a text document and the size of the shingling k, and returns an instance of the built-in data type *Set*, which ensures uniqueness of the set. To avoid collisions as much as possible, the class keeps a stateful mapping between shingling tokens and their numerical representation, defines as in incremental integer id.

```
class Shingling:

...

    def shingling(
        self,
        document: str,
        k: int
    ) -> Set[int]
```

The function *compare_sets* then uses the built-in union and intersection operations to compute the Jaccard similarity between the set representation of the documents.

```
def compare_sets(
    A: Set[int],
    B: Set[int]
) -> float
```

The *min_hash* function generates a vector of length *hash_length* by applying the min-hashing algorithm with *hash_length* random hash functions. The functions keep the universal form $h(x) = (a * x + b) \bmod N$ with N in this case being the maximum value for int32, while the parameters a and b are picked at random. The *min_hash* function converts the set representation of the document into a NumPy array of min-hashes.

```
import numpy as np

def min_hash(
    A: Set[int],
    hash_length: int = 100,
    seed: int = 0
) -> np.ndarray
```

The function *compare_signatures* leverages vectorized operations to estimate the Jaccard similarity between two documents represented by their signature and is used by the *lsh* function to calculate the approximate similarity of the documents.

```
import numpy as np

def compare_signatures(
    A: np.ndarray,
    B: np.ndarray
) -> float
```

Finally, the *lsh* function takes as input the signature matrix M representing documents, a similarity threshold t, and the number of bands to be considered. The function efficiently generates pairs of documents with a similarity greater than t, returned in a set.

```
import numpy as np

def lsh(
    M: np.ndarray,
    t: float,
    b: int = 1
) -> Set[Tuple[int, int]]
```

The function identifies candidate pairs by hashing partitions of their signatures (the bands) and considering two documents as candidate pairs when there is a collision. To keep the hashing function simple, the design choice was to simply sum the elements in the band.

2 How to run

To run the conducted experiments, follow the steps:

1. Unzip the file containing the homework.
2. Ensure to have Python 3 installed on your machine.
3. Ensure that NumPy, Itertools and Jupyter Notebook are installed in your environment.
4. Start Jupyter Notebook.
5. In Jupyter Notebook, open the notebook "Finding Similar Items" in the folder /src of the homework.
6. Press "run all".

The selection of similar items can be done by simply computing the shingling tokens for each document, comparing the resulting sets for all possible pairs of documents, and finally applying a threshold. The execution time of this operation grows exponentially with the number of documents. Therefore, an approximated set of similar items can be obtained by hashing the shingling tokens with min_hash and using the LSH algorithm.

3 Results

The experiments were performed on the dataset "Fake and real news" available on Kaggle[1]. In particular, our team compared the titles of the news to find similar documents among them.

[1] `https://www.kaggle.com/clmentbisaillon/fake-and-real-news-dataset`

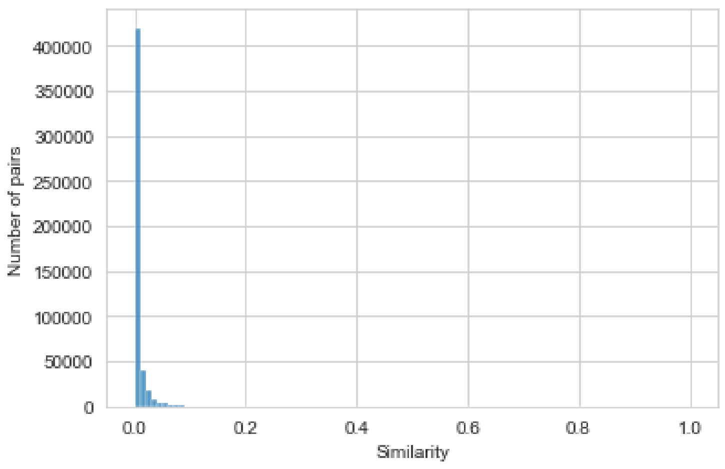

Figure 1: Distribution of similarities in the dataset.

The similarity threshold was tuned at 0.1 considering the similarity distribution in the dataset, available in Figure 1. The size of the shingles is set to 5 since the text is relatively small.

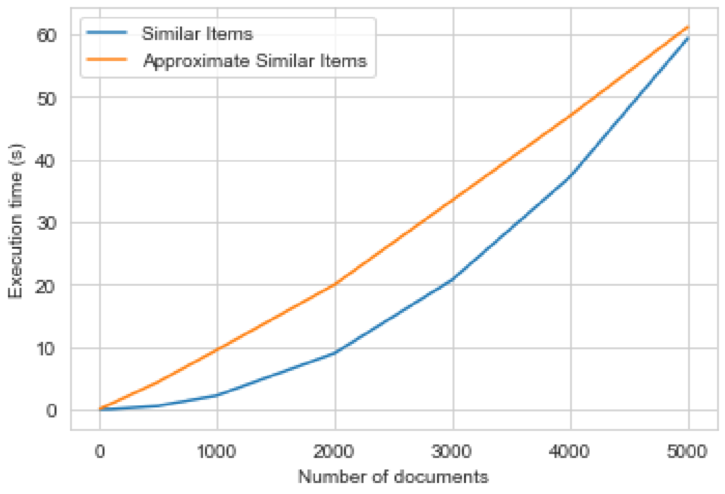

Figure 2: Comparison in the performance of LSH and exact similarity with respect to the size of the dataset.

To test the performance of the algorithms under analysis, we first compared the behavior of exact similarity versus LSH with a growing number of documents. The results are available in Figure 2. Although the exact similarity performs better with the number of documents in the test, LSH shows a linear behavior which would pay off with a larger dataset.

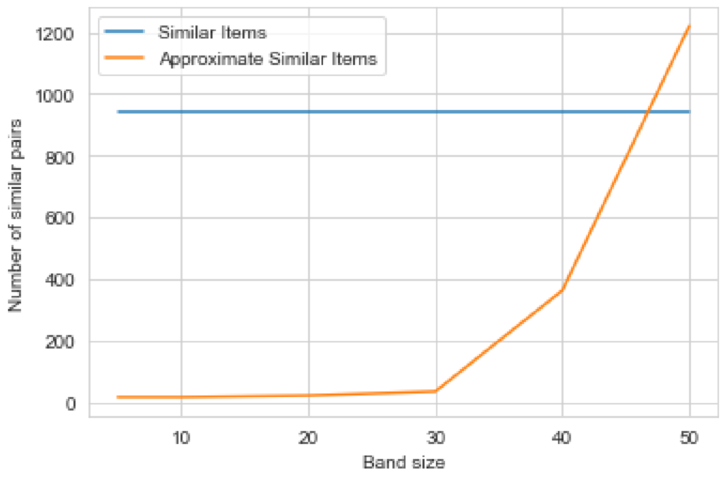

Figure 3: Tradeoff between false positives and false negatives in LSH with respect to the number of bands considered.

Moreover, in Figure 3 we show the trade-off between false positive and false negatives controlled by the number of bands considered in hashing.