Homework 1 Report - Finding Similar Items

Data Mining – ID2222 – Homework Group 23

Adriano Mundo (mundo@kth.se), Edoardo Realini (erealini@kth.se)

November 9, 2020

1. Introduction

The goal of the homework assignment 1 is to implement the stages for finding similar items, in particular textually similar documents by leveraging on three techniques: shingling, minhashing and locality-sensitive hashing (or LSH). By using all of them it is possible to find similar documents. To better explain the different phases: shingling is responsible of converting documents to set, minhashing converts large sets to short signatures while preserving similarity and locality-sensitive hashing focus on pairs of signatures that are likely to come from similar documents. The implementation has been done using Python.

2. Implementation

The code is organized in classes, each file of the zip attached with this report represents a class. For each task of the assignment there is a class with the respective functions. They are all used for running and testing the entire pipeline in the *main.py* file. In this section we will explain what each file contains and what is used for

- *Shingling.py:* it contains the class *Shingling* which is responsible for creating the k-shingles from the document. It has different functions:
  - *load_clean_document*: the function loads the document from the filename, reads it, perform the processing which consists of removing double blank spaces with one and all punctuations. Moreover, all the text is transformed to lower case.
  - *build_shingles*: the function, by taking as input the length of the shingles returns a list of the shingles for the respective Shingling object.
  - *hash_shingles*: the function hashes the character shingles obtained from the previous function.
  - *my_hash*: a custom hash function for hashing the character set of the shingles

- *CompareSets.py:* it contains the class *CompareSets* for computing the Jaccard similarity. It has only one function.
  - *calculateJaccard*: the function takes as input two sets of hashed integer and calculates the Jaccard similarity between them as the division between the intersection and union of the sets.

- *MinHashing.py:* it contains the class *MinHashing* which is responsible for building the minhash signature. It has two functions:
  - *compute_signature*: the function, by taking as input the number of hash functions for generating the permutations and the list of integer hashed shingles generates a minhash signature of the given length by following the algorithm presented in [1].
  - *hash_function*: the function hashes the shingle by using $h(x) = (ax + b) \% (2^{32} - 1)$

- *CompareSignatures.py:* it contains the class *CompareSignatures* used for calculating signature similarity. It has one function.
  - *compare_signatures:* the function, by taking two signatures as input, estimates the similarity between the two minhash signatures as a fraction of components in which they agree. This is an estimation of the Jaccard similarity between two sets [1][2].

- *LSH.py:* it contains the class LSH which is responsible for implement the locality-sensitive hashing algorithm for finding candidate pairs. It has different functions
  - *split_bands*: the function by taking the number of bands as input, returns a dictionary with key the number of documents and value the list of bands, each one of the correct row lengths.
  - *find_candidates*: the function takes as input bands and leveraging on the *split_bands* function generates the list of all candidate pairs. In this case we hash the bands and put them into buckets.
  - *filter_threshold*: the function, by having all the candidate pairs and the threshold, filters them and outputs a list of filtered pairs by similarity.
  - *find_pairs*: this is the main method of the class that given the number of bands and the threshold return the filtered pairs. These pairs are found with LSH and filtered according to the similarity threshold given as input. It leverages on the *find_candidates* and *filter_threshold* function.
  - *my_hash*: the function, by taking as input the band in form of integer list and the bucket quantity, outputs the hashed value of the band.
  - *flat_list*: a util function to flatten the list given as input

- *main.py:* this file contains all the call functions needed to test and show the entire pipeline for finding similar documents.
  It loads the file, creates the Shingling objects, build and hash the shingles with the value given through the command line (otherwise there are default values). Then it computes the Jaccard Similarity between all possible pairs of our document test set (10 news articles). After that, MinHashing objects are created and the Similarity computed. Finally, the LSH algorithm is run to find the candidate pairs of documents. In addition to the results, it also shows the time elapsed between each phase of the pipeline.

The program has been tested by using a set of 10 documents extracted from a New York Times news article dataset available on Kaggle [3].

3. How to Run the Code

In this section it is explained how to run the code provided with this short report. The recommended environment is Ubuntu >= 18.04 LTS. It is also requested to have installed on the machine Python3.

The pipeline can be run by using the following command in the directory *src.*

```
$ python3 main.py
```

It is possible to use command line arguments to set specific values for the shingles, number of hash functions for the minhash, number of bands, similarity threshold and a parameter verbose which is in charge of printing all the values respectively for the Jaccard similarity and the Signature similarity. The default configuration is the one below.

```
$ python3 main.py -ks 5 -nhf 100 -b 25 -st 0.1 -v True
```

4. Results

By running the program, it is possible to see the following output which shows each step of the pipeline correctly executed. Below, it is possible to find the output with and without the verbose option.

```
Testing and showing the LSH pipeline with parameters
Shingle dimension:  5
MinHash # functions:  100
Number of bands:  25
Similarity threshold:  0.1
Verbose:  False


Filtered couples over Threshold - Jaccard:  [(1, 5), (1, 9), (2, 3), (3, 4), (3, 8), (3,
10), (4, 8), (4, 9), (4, 10), (5, 6), (5, 8), (5, 9), (5, 10), (8, 9), (8, 10), (9, 10)]
Time Elapsed Jaccard:  0.030331850051879883


Filtered couples over Threshold - MinHashing:  [(1, 3), (1, 5), (1, 8), (1, 9), (1, 10),
(2, 3), (2, 10), (3, 5), (3, 7), (3, 8), (3, 10), (4, 10), (5, 6), (5, 7), (5, 9), (5, 10),
(7, 8), (8, 9), (8, 10), (9, 10)]
Time Elapsed MinHashing:  0.000308990478515625


Filtered couples over Threshold - LSH:  [(9, 10), (4, 10), (1, 8), (3, 10), (8, 9), (1, 9)]
Time Elapsed LSH:  0.0010175704956054688
```

```
Testing and showing the LSH pipeline with parameters
Shingle dimension:  5
MinHash # functions:  100
Number of bands:  25
Similarity threshold:  0.1
Verbose:  True


Jaccard Similarity between 1 and 2: 0.077
Jaccard Similarity between 1 and 3: 0.086
Jaccard Similarity between 1 and 4: 0.096
Jaccard Similarity between 1 and 5: 0.104
Jaccard Similarity between 1 and 6: 0.084
Jaccard Similarity between 1 and 7: 0.084
Jaccard Similarity between 1 and 8: 0.095
Jaccard Similarity between 1 and 9: 0.12
Jaccard Similarity between 1 and 10: 0.097
Jaccard Similarity between 2 and 3: 0.114
Jaccard Similarity between 2 and 4: 0.099
Jaccard Similarity between 2 and 5: 0.086
Jaccard Similarity between 2 and 6: 0.075
Jaccard Similarity between 2 and 7: 0.096
Jaccard Similarity between 2 and 8: 0.094
Jaccard Similarity between 2 and 9: 0.089
Jaccard Similarity between 2 and 10: 0.099
Jaccard Similarity between 3 and 4: 0.106
Jaccard Similarity between 3 and 5: 0.094
Jaccard Similarity between 3 and 6: 0.077
Jaccard Similarity between 3 and 7: 0.091
Jaccard Similarity between 3 and 8: 0.117
Jaccard Similarity between 3 and 9: 0.097
Jaccard Similarity between 3 and 10: 0.115
Jaccard Similarity between 4 and 5: 0.098
Jaccard Similarity between 4 and 6: 0.082
Jaccard Similarity between 4 and 7: 0.096
Jaccard Similarity between 4 and 8: 0.102
Jaccard Similarity between 4 and 9: 0.108
Jaccard Similarity between 4 and 10: 0.112
Jaccard Similarity between 5 and 6: 0.108
Jaccard Similarity between 5 and 7: 0.074
Jaccard Similarity between 5 and 8: 0.106
Jaccard Similarity between 5 and 9: 0.108
Jaccard Similarity between 5 and 10: 0.109
Jaccard Similarity between 6 and 7: 0.069
Jaccard Similarity between 6 and 8: 0.076
Jaccard Similarity between 6 and 9: 0.079
Jaccard Similarity between 6 and 10: 0.07
Jaccard Similarity between 7 and 8: 0.086
Jaccard Similarity between 7 and 9: 0.085
Jaccard Similarity between 7 and 10: 0.087
Jaccard Similarity between 8 and 9: 0.107
Jaccard Similarity between 8 and 10: 0.119
Jaccard Similarity between 9 and 10: 0.104
Filtered couples over Threshold - Jaccard:  [(1, 5), (1, 9), (2, 3), (3, 4), (3, 8),
(3, 10), (4, 8), (4, 9), (4, 10), (5, 6), (5, 8), (5, 9), (5, 10), (8, 9), (8, 10), (9,
10)]
Time Elapsed Jaccard:  0.03142380714416504
```

```
Signature Similarity between 1 and 2: 0.03
Signature Similarity between 1 and 3: 0.14
Signature Similarity between 1 and 4: 0.02
Signature Similarity between 1 and 5: 0.13
Signature Similarity between 1 and 6: 0.02
Signature Similarity between 1 and 7: 0.09
Signature Similarity between 1 and 8: 0.23
Signature Similarity between 1 and 9: 0.27
Signature Similarity between 1 and 10: 0.11
Signature Similarity between 2 and 3: 0.19
Signature Similarity between 2 and 4: 0.09
Signature Similarity between 2 and 5: 0.09
Signature Similarity between 2 and 6: 0.06
Signature Similarity between 2 and 7: 0.03
Signature Similarity between 2 and 8: 0.08
Signature Similarity between 2 and 9: 0.06
Signature Similarity between 2 and 10: 0.17
Signature Similarity between 3 and 4: 0.05
Signature Similarity between 3 and 5: 0.15
Signature Similarity between 3 and 6: 0.0
Signature Similarity between 3 and 7: 0.12
Signature Similarity between 3 and 8: 0.13
Signature Similarity between 3 and 9: 0.09
Signature Similarity between 3 and 10: 0.19
Signature Similarity between 4 and 5: 0.09
Signature Similarity between 4 and 6: 0.03
Signature Similarity between 4 and 7: 0.05
Signature Similarity between 4 and 8: 0.06
Signature Similarity between 4 and 9: 0.05
Signature Similarity between 4 and 10: 0.16
Signature Similarity between 5 and 6: 0.1
Signature Similarity between 5 and 7: 0.12
Signature Similarity between 5 and 8: 0.09
Signature Similarity between 5 and 9: 0.1
Signature Similarity between 5 and 10: 0.17
Signature Similarity between 6 and 7: 0.01
Signature Similarity between 6 and 8: 0.0
Signature Similarity between 6 and 9: 0.01
Signature Similarity between 6 and 10: 0.02
Signature Similarity between 7 and 8: 0.1
Signature Similarity between 7 and 9: 0.02
Signature Similarity between 7 and 10: 0.09
Signature Similarity between 8 and 9: 0.19
Signature Similarity between 8 and 10: 0.15
Signature Similarity between 9 and 10: 0.15
Filtered couples over Threshold - MinHashing:  [(1, 3), (1, 5), (1, 8), (1, 9), (1, 10),
(2, 3), (2, 10), (3, 5), (3, 7), (3, 8), (3, 10), (4, 10), (5, 6), (5, 7), (5, 9), (5, 10),
(7, 8), (8, 9), (8, 10), (9, 10)]
Time Elapsed MinHashing:  0.00036978721618652344


Filtered couples over Threshold - LSH:  [(9, 10), (4, 10), (1, 8), (3, 10), (8, 9), (1, 9)]
Time Elapsed LSH:  0.0010228157043457031
```

References

[1] Chapter 3 in *Mining of Massive Datasets,* by Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman, 2nd edition, Cambridge University Press, 2014

[2] Finding Similar Items lecture, ID2222, Data Mining, KTH Royal Institute of Technology

[3] New York Times Articles dataset, available on: https://www.kaggle.com/nzalake52/new-york-times-articles