Homework 1: Finding Similar Items: Textually Similar Documents

Group 28: Junjie shan, Yuxin Meng

December 2021

1 Short description

This is the report for data mining homework 1, finding similar items: textually similar documents, which is based on Python. We implemented Shingling, CompareSets, MinHashing, CompareSignatures and LSH as illustrated in the homework description. For the dataset, we used some online posts text to test our implementation.

2 Instructions how to build and to run

2.1 Data preprocess

The dataset are stored in the 'dataset' folder in the format of .json files. In our implementation, we built a DataReader class to read all .json files in the dataset folder, and take the 'text' area, which is used to store the content of posts. Then, we built a Preprocessor class to process the data read using DataReader, including converting them to lowercase letters, removing punctuation, etc.

2.2 Shingling

We built a Shingling class including four functions, which takes the preprocessed data and picked shingling size k to return the shinglings in each document and all documents chosen, then we convert them into hash values.

2.3 CompareSets

In this part, we construct a CompareSets class to compare the Jaccard similarity between two sets of shingling. The Jaccard similarity is computed as intersection of two sets divided by union of two sets.

2.4 MinHashing

The class MinHashing contains three functions. The first is a definition of a hash function with the structure (ax + b) mod c, where c is a very large prime number and a and b are random integers in the range 1 to c-1. The second function calculates the signature of all documents, and here we call the third function we have implemented, which calculates the minhash value for each single document and then obtains the document's signature.

2.5 CompareSignatures

Comparing the similarity of two signatures only requires finding the number of elements that are the same in both signatures, this is because the signatures being compared are of the same length. So the similarity of the signatures is equal to the number of identical elements divided by the number of elements contained in the signature.

2.6 LSH

In the class LSH, we implemented to find possible candidate pairs based on the input signature matrix and threshold. Firstly, the LSH implementation requires parameters about the number of bands and the number of raws, which we calculate based on the threshold, as stated in the lecture sildes, page 57. When the formula is closest to the value of threshold, the ideal parameters are obtained so that both false negatives and false positives are in the right range. Next, the lsh function implements exactly how to find candidate pairs. The signature matrix will be divided, as our signature is a list, so it needs to be converted into an array, each row in the two-dimensional array is the signature of each document, so we divide it by column. Each row in each chunk is hashed and mapped to a bucket, and if it is mapped to the same bucket, then it is likely to be a similar pair. Do the same for each band, and finally remove duplicate candidate pairs.

2.7 Run

This can be simply run with **python3 hw1.py**, since it is totally based on Python3.

3 Results

3.1 Shingling

This part of the hashed shinglings of our first file:

```
{-9217207221531522940,
 -9109013801460705256,
 -9057603136448146073,
 -9056945709803061466,
 -8979539566370241928,
 -8970535010280145245,
 -8929164096022720059,
 -8912305383278475014,
 -8883150859208523930,
 -8855046193399427317,
 -8839048470470613262,
 -8801661373800885149,
 -8796806484455308682,
 -8674721097173465307,
 -8658662684830016865,
```

Figure 1: Hashed shinglings

3.2 CompareSets

This is the Jaccard similarities of the first ten files in our data set:

```
Jaccard Similarity Matrix of Dataset
[[1.    0.642 0.441 0.017 0.02  0.023 0.105 0.016 0.027 0.117]
 [0.642 1.    0.446 0.023 0.01  0.028 0.101 0.016 0.046 0.113]
 [0.441 0.446 1.    0.009 0.007 0.019 0.108 0.01  0.022 0.119]
 [0.017 0.023 0.009 1.    0.021 0.155 0.019 0.042 0.166 0.019]
 [0.02  0.01  0.007 0.021 1.    0.    0.507 0.005 0.011 0.009]
 [0.023 0.028 0.019 0.155 0.    1.    0.    0.283 0.177 0.158]
 [0.105 0.101 0.108 0.019 0.507 0.    1.    0.003 0.095 0.012]
 [0.016 0.016 0.01  0.042 0.005 0.283 0.003 1.    0.06  0.399]
 [0.027 0.046 0.022 0.166 0.011 0.177 0.095 0.06  1.    0.024]
 [0.117 0.113 0.119 0.019 0.009 0.158 0.012 0.399 0.024 1.   ]]
```

Figure 2: Jaccard Similarity

3.3 MinHashing

This is part of the MinHash signatures of our first file:

```
[356216,
 2656635,
 70295,
 2783973,
 1060452,
 3125580,
 1332334,
 1068744,
 2662547,
 3593569,
 2402521,
 6005756,
 2343869,
 1443029,
 563859,
 76812,
 1985779,
```

Figure 3: MinHash Signatures

3.4 CompareSignatures

This is the similarities of the signatures of the first ten files of our dataset:

```
Signature Similarity Matrix of Dataset
[[1.    0.646 0.462 0.014 0.004 0.026 0.112 0.018 0.032 0.11 ]
 [0.646 1.    0.462 0.016 0.002 0.024 0.12  0.018 0.038 0.102]
 [0.462 0.462 1.    0.014 0.002 0.022 0.118 0.012 0.028 0.108]
 [0.014 0.016 0.014 1.    0.02  0.166 0.02  0.05  0.18  0.022]
 [0.004 0.002 0.002 0.02  1.    0.    0.508 0.01  0.016 0.01 ]
 [0.026 0.024 0.022 0.166 0.    1.    0.    0.294 0.158 0.172]
 [0.112 0.12  0.118 0.02  0.508 0.    1.    0.01  0.078 0.02 ]
 [0.018 0.018 0.012 0.05  0.01  0.294 0.01  1.    0.06  0.412]
 [0.032 0.038 0.028 0.18  0.016 0.158 0.078 0.06  1.    0.03 ]
 [0.11  0.102 0.108 0.022 0.01  0.172 0.02  0.412 0.03  1.   ]]
```

Figure 4: Signature Similarity

3.5 LSH

This is the candidates found in the first ten files of our dataset, followed by the Jaccard similarity and Signature similarity:

```
Candidate pair(4, 6)
0.5074183976261127
0.508
Candidate pair(0, 2)
0.4412811387900356
0.462
Candidate pair(0, 1)
0.6416819012797075
0.646
Candidate pair(1, 2)
0.445993031358885
0.462
Runtime of was 4.49600076675415
```

Figure 5: Candidates Pairs