ID2222 Data Mining Homework 1

Finding Similar Items

Luís Santos `lmpss@kth.se`

Kevin Dalla Torre `kevindt@kth.se`

November 2020

1 Step 1

For step 1 we have a class Shingling that by default considers 10-shingles. This can be changed when initializing the object. This class has a method `hash_shingles` that hashes the shingles to an integer. It does so by calculating the following:

$$(\sum_{i=1}^{K} 100^i * unicode(shingle[i])) \mod 2^{32} \tag{1}$$

The method has the argument shingle which is the shingle that is going to be hashed. K is the shingle length (10 by default). The index or position of a character in a shingle is presented as i in the equation. For each character the method takes the Unicode of that character and multiplies it with 100^i It then calculates (the sum of these values) mod 2^{32}. The value 2^{32} is taken from the book. There it was for 9-shingels but we made the assumption that it would work for 10-shingles as well.

Then there is the method `create_unique_shingles` that has a document as an input then iterates over all the shingles in that document and hash them using the method described above and adds the values to a list. After that the method removes duplicates and sort the list. The singles them self are created by taking the sequence of characters in a document with the positions ranging from i to K+i. This is done for all i ranging from the first position to (document length) - K , (range(document length - K)).

2 Step 2

For this step we have a class CompareSets that contains the method `jaccard_similarity`. This method takes two collections of shingles as arguments. The method computes the jaccard similarity by taking the intersection of the collections and then dividing by their union.

$$\frac{intersection}{union} \tag{2}$$

3 Step 3

For the third step we have a class MinHashing which implements the two algorithms discussed in the book for computing a minhash signature. One method uses permutations and the other one a hash function over the rows. Both methods receive as input a characteristic matrix. Importantly, since we know a priori such a matrix is likely sparse, we use the sparse matrix data structure from the `scipy` library. The method `compute_signature_hash` creates the illusion of a permutation over the rows via a hash function. We apply the following hash function from the family of universal hash functions, where x signifies the row index, a is a random odd number between 1 and p, b is a random number between 0 and p, and to decrease hash collisions p is a prime number larger than m. Finally, m is the length of the signature.

$$h(x, a, b) = (ax + b \mod p) \mod m \tag{3}$$

The method `compute_signature_perm` takes N permutations of the characteristic matrix's rows, where N is the length of the signature, and it records the positions of the initial ones in a column-wise fashion to compute a minhash signature. In our tests, the hash version is faster than the permutation method, and their signatures' similarity tend to differ over a sizeable margin.

4 Step 4

The solution to this problem is a class CompareSignatures that has one method named `signature_similarity`. The method takes the signature with the shape (number of hash functions, number of documents) and the

index of the documents (columns) being compared. Then with the help of the np.mean function it calculates the following:

$$\sum_{i=1}^{N} \frac{I(signature[i, doc_1]) = signature[i, doc_2])}{N} \tag{4}$$

This is the probability that both documents have the same value in a given row (if we pick the row at random). This approximates the jaccard similarity. The approximation becomes better as $N \to \infty$. I in equation 4 is an indicator function.

5 Step 5

$$n = b * r \implies r = \frac{n}{b} \tag{5}$$

$$(\frac{1}{b})^{\frac{1}{r}} \approx t \tag{6}$$

$$(5) + (6) \implies (\frac{1}{b})^{\frac{1}{r}} = (\frac{1}{b})^{\frac{b}{n}} = t_{calc} \approx t \tag{7}$$

Here we have combined both equations to eliminate r. Since n is the row length of the signature matrix and thereby a fixed value we can see how t depends on b (independent from r). We can calculate t_{calc} for different values of b. We then choose the b that gives us t_{calc} that has the values closest to the t value desired. r is then determined by the equation $r = n/b$.
We have implemented a class LSH with a method `find_candidates` which computes all candidate pairs using the LSH technique. We apply banding over the signature matrix, whereby b is a parameter that we pass in the initialization, and inside each band we hash each column and store the document IDs in the hash table. In Python, we use dictionaries as hash tables, and we convert a vector to a hashable representation via the built-in tuple function. We use different hash tables for each band and compute all pairwise combinations for the candidates. Afterwards, we have a method `find_similar` which assesses for each candidate pair if the signature similarity is above a certain threshold, and if so we retrieve those documents as similar documents.

6 How to run

Our homework solution can be run via the command `python run_hw.py`. Check Figure 1 for the command-line parameters, as well as a short description of the functionality of each parameter. Check Figure 2 to observe an example run, as well as to obtain the default values of the parameters. The datasets can be downloaded from `https://webhose.io/free-datasets/`. In our tests, we experimented with two webhose datasets, namely the Covid discussions and Covid blog posts datasets. To run our Python script, download a webhose dataset and move the original archive to a `datasets/` folder.

```
(data-mining) luis@luis-PC:~/kth/data mining/id2222-data-mining/homework 1$ python run_hw.py --help
usage: run_hw.py [-h] [-dataset-file DATASET_FILE] [-n-documents N_DOCUMENTS]
                 [-k-shingles K_SHINGLES] [-n-signature N_SIGNATURE]
                 [-n-bands N_BANDS] [-sim-threshold SIM_THRESHOLD]
                 [-use-permutations]

Find similar documents using minhashing and LSH techniques.

optional arguments:
  -h, --help            show this help message and exit
  -dataset-file DATASET_FILE
                        path to a webhose dataset
  -n-documents N_DOCUMENTS
                        number of documents to read from dataset
  -k-shingles K_SHINGLES
                        construct shingles of character length k
  -n-signature N_SIGNATURE
                        build a minhash signature of length n
  -n-bands N_BANDS      number of bands for locality-sensitive hashing
  -sim-threshold SIM_THRESHOLD
                        similarity threshold for retrieving documents
  -use-permutations     compute minhashing via permutations or hashing
(data-mining) luis@luis-PC:~/kth/data mining/id2222-data-mining/homework 1$
```

Figure 1: Command-line parameters of our solution.

```
(data-mining) luis@luis-PC:~/kth/data mining/id2222-data-mining/homework 1$ python run_hw.py
Namespace(dataset_file='covid-blog-posts.zip', k_shingles=10, n_bands=100, n_documents=100, n_signature=500, sim_threshold=0.8, use_permutations=False)
similar documents: [(25, 98), (39, 73)]
(data-mining) luis@luis-PC:~/kth/data mining/id2222-data-mining/homework 1$
```

Figure 2: Default values Command-line parameters of our solution.