Karl M. Heuer / David E. Roberson

Autumn 2024
Exercise class:
in 302 on September 24th

Exercises for Computational Tools for Data Science (02807)

Week 4: Similar Items

References and Reading

1. Chapter 3 of Mining of Massive Data Sets, Jure Leskovec, Anand Rajaraman, and Jeff Ullman.

Exercise 1: Setup

Download the test data and template file `similarity.py`. Also install the `mmh3` library. See https://pypi.org/project/mmh3/

Exercise 2: q-shingles

Implement a function `shingle` that takes an integer q and a string and produces a list of shingles, where each shingle is a list of q words.

Solution: Here, the input string (a document in the case of the test files) would consist of separate words. The q-shingles for this string would be all of the sequences of q consecutive words in the string. For example, if the string was '`i want to go home do you want to`' and $q = 2$, then the q-shingles would be {`i want`, `want to`, `to go`, `go home`, `home do`, `do you`, `you want`}. Note that we do not include '`want to`' twice because we only care about the *set* of shingles, we do not care about the order. Of course it will probably be output as a list which does have an order but there should not be repeats. See Section 3.2.2 (page 78) in the online version of the book. Note that in the book they use sequences of q *characters* rather than words, but the principle is the same.

Exercise 3: Minhashing Solve the following exercises.

3.1 Implement a minhash algorithm `minhash` that takes a list of shingles and a seed for the hash function mapping the shingles, and outputs the minhash. Feel free to use the `listhash` function in the template.

3.2 Extend the minhash algorithm to output k different minhashes in an array. Use different seeds for each minhash, e.g., $1, \dots, k$.

Solution: Given a hash function h that maps to integers, the minhash $h_{\min}$ of a set of elements S is the minimum value that the function h takes on the set S, i.e., $h_{\min}(S) = \min\{h(s) : s \in S\}$. See Section 3.3.3 (page 82) in the book. In this case the exercise suggests to use the `listhash` function provided. So the `minhash` function should take a list of shingles (and a seed for `listhash`) and compute the minhash for this list, i.e., it should compute `listhash` of every shingle in the list and return the minimum value. For the second part just add k as input and have it return a sequence of k minhashes each using a different seed for the hash function. Note that we care about the order of the minhashes here, because we will be comparing the values of the i^{th} minhash of a given list of shingles with the i^{th} minhash of another list of shingles.

Exercise 4: Signatures

Construct a function `signatures` that takes the `docs` dictionary and outputs a new dictionary consisting of document id's as keys and signatures as values.

Solution: This should take each document and convert it to a list of q-shingles (so q should be an input to the function) and then use the function from 3.2 to convert that list of shingles for that document to a sequence of k minhash values (so k will also be an input) which is known as the *signature* of that document. See Section 3.3.4 (page 83) for info on minhash signatures.

Note that it is not necessarily most efficient to simply compute the signature for each document separately. In the book in Section 3.3.5 (page 84), they give an algorithm for computing the signatures of a collection of sets/documents simultaneously. I give my attempt at pseudocode for this algorithm below. Let U be the set of all shingles produced from a given set D of documents and let shingle(d) denote the list of shingles produced from a given document $d \in D$ (assume that these have already been computed prior to running the below code). Suppose we are creating signatures using minhashes from hash functions $h_1, \ldots, h_k$. The following creates a matrix SIG with rows indexed by $1, \ldots, k$, and columns indexed by D such that the d column is the signature of document d.

Algorithm 1 MakeSignatures

```
1: Initialize all $\mathrm{SIG}(i,d) = \infty$
2: for $s \in U$ do
3:     Compute $h_1(s), \ldots, h_k(s)$
4:     for $d \in D$ do
5:         if $s \in \mathrm{shingle}(d)$ then
6:             for $i \in \{1, \ldots, k\}$ do
7:                 $\mathrm{SIG}(i,d) \leftarrow \min\{h_i(s), \mathrm{SIG}(i,d)\}$
```

Note that after the above finishes the value of SIG(i, d) will be the minhash of shingle(d) with respect to the hash function h_i.

Exercise 5: Jaccard Similarity

Implement a function `jaccard` that takes two document names and outputs the estimated Jaccard similarity using signatures.

Solution: Given two sets S and T, the Jaccard similarity of S and T is equal to $|S \cap T|/|S \cup T|$, i.e., the size of the intersection divided by the size of the union of the two sets (see Section 3.1.1, page 74). For a random *injective* hash function h, the probability that $h_{\min}(S) = h_{\min}(T)$ is equal to the Jaccard similarity of S and T (see Section 3.3.3, page 83). The hash functions used in practice won't necessarily be injective, but assuming there are not many collisions we can still use this as an estimation of Jaccard similarity. So this function should take two documents, convert each to a list of q-shingles (so q should be an input) and then use the function from 3.2 to convert these into signatures consisting of k minhash values (so k will be an input). Then it estimates the Jaccard similarity of the two documents by counting how many positions of the two signatures agree and then dividing that by k.

Exercise 6: Find Similar Items

Implement a function `similar` that finds all pairs of documents whose estimated Jaccard similarity is ≥ 0.6. Test your program for different values of k and q. Compare your results for most similar documents with your own visual impression of the similarity of files.

Solution: This just combines techniques from previous questions. You can use the function from Exercise 4 to convert all the documents to signatures, and then write a function that computes Jaccard similarity based on signatures (essentially the second half of the function from Exercise 5). Then compute/estimate the Jaccard similarity for all pairs of documents and return those with Jaccard similarity at least 0.6.

Exercise 7: Locality-Sensitive Hashing

Use locality-sensitive hashing to speed up your solution to the find similar item exercise.

Solution: Here, we take our signatures for all of our documents and break them up into b blocks of length r (so $k = br$ where k is the length of the signatures). For each block, we apply a hash function to that part of the signature for each of the documents, and if two documents get hashed to the same value for at least one of the blocks, then they will be considered a candidate for being similar. We then compute the Jaccard similarity of only the candidate pairs using their full signatures. Note that the values of b and r need to chosen correctly so that potential similar pairs are correctly identified. If one wants to identify pairs with Jaccard similarity at least some value s, then one should choose b and r such that $(1/b)^{1/r} \approx s$. See Section 3.4. (page 91) in the book.