Programming Assignment 4

Department of Computer Science, University of Wisconsin – Whitewater
Algorithms in the Real World (CS 738)

Instructions For Submissions

- Submission is via Canvas as a single zip file.
- Submit code and a brief report.
- No need to include the algorithm description in the report.
- Make sure you cite all resources used. If you use online code resources, please make sure that it is not a verbatim copy.

---

1 Data Description and Testing

We are going to check documents for plagiarism using a MinHash. The data is provided in the assignment folder `Data`. The data comprises of 4 collections of labeled articles – `articles100`, `articles1000`, `articles2500`, and `articles10000`. For each set of articles, the labeled plagiarized articles are listed in the corresponding `.truth` files. Additionally, you are given a set of unlabeled documents – `setA`, `setB`, `setC`, `setD`, and `setE`. Each folder contains a file `JaccardSimilarity.csv` that contains estimated (based on MinHash) Jaccard Similarity scores for each pair of files (which depends on the primes, and so it may be different in your tests).

2 Prime Generation

You do not need to write any code for this part, but I believe there is a very important algorithm to learn over here, which is known as Miller-Rabin primality test. We have used this test to generate large primes have been generated in the following way:

- The input asks for the number of primes desired and what is the maximum size of the prime (in bits). This method generates $4 * numPrimes$ many primes, each having at most $maxBits$ in its binary representation. Then, the method returns $numPrimes$ random primes among the generated ones.
- Starting from $2^{maxBits} - 1$, we go down two numbers at a time and collect the first $4 * numPrimes$ along the way.
- To test a number n for primality, we use the following strategy. If $n \leq 10000$, we just use the standard $O(\sqrt{n})$ time algorithm to determine if the number is prime. If $n > 10000$, we first use Rabin-Miller test. If the test returns $false$, then the number is surely composite. If the test returns $true$, then we deterministically check for primaility using the $O(\sqrt{n})$ time algorithm. The use of Rabin-Miller makes the process faster as we can eliminate most non-primes quickly.

3 Task 1: HashCode

The hash-code a string S of length n for a given prime is computed as:

$$\left(2^{n-1} \cdot S[0] + 2^{n-2} \cdot S[1] + 2^{n-3} \cdot S[2] + \cdots + 2 \cdot S[1] + S[0]\right) \ \% \ prime$$

To compute the hash-code, do the following:

- Initialize $hashCode = 0$
- Now run a loop over the string and update $hashCode = \big(2 * hashCode + S[i]\big)\%prime$

4 Task 2: MinHashCode

Say, the words array is $\big[w_0, w_1, w_2, \ldots, w_{n-1}\big]$. We will compute a single MinHashCode for the words array as follows.

- Starting from w_0, combine $shingles$ number of words at a time. For example, if $shingles = 3$, then the concatenated words are $\big\{w_0w_1w_2, w_1w_2w_3, w_2w_3w_4, \ldots, w_{n-4}w_{n-3}w_{n-2}, w_{n-3}w_{n-2}w_{n-1}\big\}$
- For each concatenated shingle, compute the hash-code using Task 1 for the given prime.
- Pick the minimum among all these hash codes and return it.

5 Task 3: Plagiarism Detection

- Start by creating an integer two dimensional dynamic array $allMinHashCodes$. This is vector of integer vectors in C++ or an ArrayList of integer ArrayLists in Java.
- Create a string dynamic array called $docIDs$
- First obtain all the `.txt` files in the folder given by $folderPath$
- Now, for each file in the folder, do the following (this is a simplified way of expressing the main idea; you can and probably should do it more space/time efficiently):
  - Read the file one character at a time into a string *content*. While reading the content, do the following:
    - If the character is a newline character, append a space to content.
    - We are only interested in English letters, numbers, and spaces. So, if the character is anything else, then do not append it to *content*. Otherwise, append the character to *content*.
  - At this point *content* only contains the English letters, numbers, and spaces. Convert all upper-case letters to lower-case.

    Ideally, at this point we should remove articles, connecting verbs (such as *is*, *are*, etc), and stem the words (remove *ing*, *ly*, etc from the end). If you are interested, you can do it. However, for the given data set, I did not find it necessary.
  - Split *content* into a string array *words*[ ] by using space as the delimiter.
  - Create an integer dynamic array $minHashCodesInThisDoc$

  - For each prime number in the array $primes[\ ]$, compute the MinHashCode of $words[\ ]$ using Task 2, and then add the code to $minHashCodesInThisDoc$
  - Add $minHashCodesInThisDoc$ to $allMinHashCodes$
  - Add the name of the file to $docIDs$

- Create a hash-set (HashSet in Java and unoredered_set in C++) called *codes*.
- For each document pair (i, j), where i and j are indexes over the number of documents (given by the size of *documents*), compute their Jaccard Similarity as follows $\big($note that you have to only compute once for a pair; so, j can start at $i + 1\big)$:
  - Add the min-hash-codes for document i (which are stored in index i of $allMinHashCodes$) into the hash-set *codes*.
  - Loop over row j of $allMinHashCodes$, and count how many of them are already present in *codes*. Let the number of matches be k.

    Estimate Jaccard Similarity as $k/(number\ of\ primes)$
  - Report (i, j) as a plagiarized pair if the Jaccard Similarity exceeds the threshold.
  - Clear the hash-set before moving onto the next pair.

6 Task 4: Report

A skeleton code has been provided in the `SkeletonCode` folder for both C++ and Java. Code for prime generation has been given. Snippets have been written in the `Test` file for testing each of the following sections, code for which you have written in the previous three tasks. You will also note that I have used twenty randomly generated 32-bit primes (as described in the prime generation section), and 3-shingles.

Write a very brief report mentioning the document pairs (for each of the data sets, both *labeled* and *unlabeled*) that have been found to be plagiarized for each of the following threshold scores: 0.3, 0.5, 0.7, and 0.9.