Com S 435/535 Programming Assignment 2
600 Points
Due: Oct 25, 11:59PM
Late Submission Due Oct 26, 11:59PM (25 % Penalty)

In this programming assignment, you will implement minhash and locality sensitive hashing to estimate Jaccard similarity among documents and to identify near-duplicate documents.

Note that the description of a programming assignment is not a linear narrative and may require multiple readings before things start to click. You are encouraged to consult instructor/Teaching Assistant for any questions/clarifications regarding the assignment.

Your programs must be in Java, preferably Java 8.1. **You are allowed to work in groups of size 2**. However, **only one submission per group please**. Please do not forget to read the guidelines before you start implementation.

1 Jaccard Similarity of Multisets

In a multiset an element can appear multiple times. For example $\{1, 3, 1, 2, 3, 4\}$ is a multiset. In lectures, we considered *binary Jaccard Similarity* between documents. For this we viewed each document as a *set* of terms (equivalently binary vector) and used Jaccard Similarity between sets to estimate similarity of documents. A downside of this is that it ignores *term frequency*. Consider two documents A and B. Suppose in A term 1 appears once term 2 appears 4 times, and in B term 1 appears once and terms 2 appears once. If we use binary Jaccard similarity, then the set of terms corresponding to A is $\{1, 2\}$ and set of terms corresponding to B is $\{1, 2\}$ thus similarity is 1. This is bit unsatisfactory.

A way to get around this is to view each document as a vector that keeps track of frequencies and use Jaccard similarity between (non binary) vectors. However, we do not know how to do minhash and LSH this similarity measure. So we resort to a different method, we view document as a multiset and use jaccard similarity between multisets.

Let A and B be two multisets. Let U be the *set* of terms that appear in $A \cup B$. Note that U is a normal set, **not a multiset**. Given x, let $f_A(x)$ be the number of times x appears in A and let $f_B(x)$ be the number of times x appears in B. Now

$$|A \cap B| = \sum_{x \in U} \min\{f_A(x), f_B(x)\},$$

and

$$|A \cup B| = \sum_{x \in U} \max\{f_A(x), f_B(x)\},$$

Now *MultiSet Jaccard Similarity* A and B is

$$MuliSetJaccardSimilarity(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

In this PA, you will view documents as mutisets and use Multiset Jaccard Similarity as the similarity measure. From now, we will use the *Jaccard Similarity* in place of Multiset Jaccard Similarity.

2 Document Preprocessing

Note that *term* of a document could be a word or a shingle. For this PA, we take *word* as *term*. Do (ONLY) the following preprocessing before extracting terms: Convert all characters into lower case, remove (ONLY) the following punctuation symbols: Period, Comma, Colon, Semi Colon, apostrophe. remove all words of lengths one and two, and remove the word “the”.

3 MinHash

Your first task is to design a class named `MinHash`. This class should have following constructors and methods.
`MinHash(String folder, int numPermutations).` `folder` is the name of a folder containing our document collection for which we wish to construct MinHash matrix. `numPermutations` denotes the number of permutations to be used in creating the MinHash matrix.

`allDocs` Returns an array of `String` consisting of all the names of files in the document collection.

`minHashMatrix()` Returns the MinHash Matrix of the collection. Return type is 2D array of ints.

`termDocumentMatrix()` Return the term document matrix of the collection. Return type is 2D array of ints.
`numTerms()` Returns the size of union of all documents (after preprocessing). Note that each document is a multiset of term.

`numPermutations` Returns the number of permutations used to construct the MinHash matrix.

3.1 MinHashSimilarities

This will will invoke `MinHash` to compute/estimate similarities. This class will have following constructor and methods.
`MinHashSimilarities(String folder, int numPermutations).` Creates an instance of `MinHash`, and calls the methods `termDocumentMatrix` and `minHashMatrix` to store the respective matrices.

`exactJaccard(String file1, String file2)` Gets names of two files (in the document collection) `file1` and `file2` as parameters and returns the exact Jaccard Similarity of the files. Use the `termDocumentMatrix` computed (by the constructor) for this. Return type is double.

`approximateJaccard(String file1, String file2)` Estimates and returns the Jaccard similarity of documents `file1` and `file2` by comparing the MinHash signatures of `file1` and `file2`. Use the `MinHashMatrix` computed by the constructor. Return type is double.

`minHashSig(String fileName)` Returns the MinHash the minhash signature of the document named `fileName`. Return type is an array of ints.

3.2 MinHashAccuracy

Create a class named `MinHashAccuracy` that tests the how accurately MinHash matrix can be used to estimate Jaccard Similarity. This class should have a method named `accuracy` that does the following:

- Gets name of a folder, number of permutations to be used and an error parameter (less than one) as parameters (in that order). Let us use ϵ to denote the error parameter. This method will create an instance of `MinHashSImilarities`.
- For every pair of files in the document collection, compute exact Jaccard Similarity and approximate Jaccard similarity.
- Reports the number of pairs for which exact and approximate similarities differ by more then ϵ.

3.3 MinHashTime

This class tests whether it is faster to estimate Jaccard Similarities using MinHash matrix. This class has a method named `timer` that

- Gets name of a folder, number of permutations to be used as parameters, and creates an instance of `MinHashSimilarities`. Report the the time taken to construct an instance of `MinHashSimilarities`.
- For every pair of files in the folder compute the exact Jaccard Similarity ; Report the time taken (in seconds) for this task.
- Compute the MinHashMatrix and use this matrix to estimate Jaccard Similarity of every pair of documents in the collection. Report the time taken for this task.

4 LSH

This class implements locality sensitive hashing to detect near duplicates of a document. Recall that given a $K \times N$ MinHash matrix M, we perform locality sensitive hashing as follows. For a given b, divide the rows of M into b bands each consisting of $r = k/b$ rows. Create b hash tables $T_1, \cdots, T_b$. For each document D_i let sig be its MinHash signature which is an array of k integers. Divide sig into b bands (each band has r entries), and compute hash value of each band. If jth band of sig is hashed to t, then store the document (name) D_i at $T_j[t]$.

This class should have following methods and constructors.

`public LSH(int[][] minHashMatrix, String[] docNames, int bands)` Constructs an instance of `LSH`, where `docNames` is an array of Strings consisting of names of documents/files in the document collection, and `minHashMatrix` is the MinHash matrix of the document collection and `bands` is the number of bands to be used to perform locality sensitive hashing. You may assume that the ith column of `minHashMatrix` is the minhash signature of `docNames[i]`.

`nearDuplicatesOf(String docName)` Takes name of a document `docName` as parameter and returns an array list of names of the near duplicate documents. Return type is ArrayList of Strings

4.1 NearDuplicates

This class puts together `MinHash` and `LSH` to detect near duplicates in a document collection. This class should have a constructor named `NearDuplicates` that gets the following information as parameters (in the prescribed order):

- Name of the folder containing documents
- Number of Permutations to be used for MinHash
- Similarity threshold s, which is a double

Constructor should create an instance of `LSH`. In addition this class will have a method named named `nearDuplciateDetector` that gets (name of) a document as parameter, which is a string. Then this method returns a list of documents that are at least s-similar to $docName$, by calling the method `nearDuplicatesOf` from LSH. Note that this list may contain some `False Positives` —Documents that are less than s-similar to $docName$. DO NOT eliminate false positives.

5 Additional Classes

If it helps, you may write additional classes and methods.

6 Report

Write a brief report that includes the following.

For the class `MinHash`:

- Your procedure to collect all terms of the documents and the data structure used for this.
- Your procedure to assign an integer to each term.
- The permutations used, and the process used to generate random permutations.

For the class `MinHashAccuracy`:

Run the program with following choices of `NumPermutations`: 400, 600, 800 and following choices for ϵ: 0.04, 0.07, 0.09. Note that you have nine possible combinations. Run the program on the files from `space.zip`.

Report the number of pairs for which approximate and exact similarities differ by more than ϵ for each combination. What can you conclude from these numbers?

For the class `MinHashTime`:

Use 600 permutations on files from `space.zip`. Report the the total run time to calculate exact Jaccard similarities and approximate Jaccard similarities (between all possible pairs).

Finally, run `nearDuplicateDetector` on the files from `PA2.zip` (with at least two choices of s). Run the program on at least 10 different inputs For each input: List all the files that are returned as near duplicates in your report.

7 Suggestions

Before you start, please make sure you understand the notions of (multiset) Jaccard Similarity, random permutations, MinHash signature, minhash matrix and locality sensitive hashing. For minhash, we need a random permutation. There are two choices, randomly create k permutations and store them all. You may also use $(ax + b)\%p$ as your permutation (for an appropriate choice of a prime number p).

8 Data

The zip file `space.zip` contains around thousand articles from news group `sci.space`.

The zip file `articles.zip` contains around 3 thousand articles from news groups `sci.space`, `sports.baseball`, `sports.hockey`.

For each file that appears in `articles.zip`, I created 7 near duplicate documents (similarity around 0.96). These files are in `PA2.zip`. Take a look at the names of the files. For example, for file `space-0.txt`, `space-0.txt.copy1`, `space-0.txt.copy2`, $\cdots$, `space-0.txt.copy7` are near duplicates.

So if you run `NearDuplicates` with input `space-0.txt` and 0.9 as similarity threshold (for appropriate choice of number of permutations and bands), then the program should minimally output all of `space-0.txt.copy1`, `space-0.txt.copy2`, $\cdots$, `space-0.txt.copy7` as near duplicates.

9 Guidelines

You are allowed to work in groups of two and are allowed to discuss with other groups. However, I strongly suggest that you think about the problems on your own before discussing. However, your group should write your programs and the report, without consulting members of other group. In your report you should acknowledge the students with whom you discussed and any online resources that you consulted. This will not affect your grade. Failure to acknowledge is considered *academic dishonesty*, and it will affect your grade.

10 What to Submit

Please submit all .java files and the report via Canvas. Your report should be in pdf format (include both the team numbers names in the report). Please do not submit .class files. Please DO NOT submit data files. Please zip all .java files and the report, name the file `PA2YourUserID.zip`. **Only one submission per group please.**

Have Fun!