Com S 435X/535X Programming Assignment 2
600 Points
Due: Mar 10, 11:59PM
Late Submission Due: Mar 11, 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.

1 MinHash

Recall that given a collection of documents $D = \{D_1, \cdots D_N\}$, consisting of terms $\{t_1, \cdots, t_M\}$, the term document matrix is an $M \times N$ matrix and this matrix contains all the information to compute Jaccard similarity of any two documents. In lectures, we have seen how to construct a $K \times N$ minhash matrix that can be used to estimate similarity of any two documents, where K is the number of random permutations.

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.

`exactJaccard(String file1, String file2)` Get names of two files (in the document collection) `file1` and `file2` as parameters and returns the exact Jaccard Similarity of the files.

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

`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`.

`minHashMatrix()` Returns the MinHash Matrix of the collection.

`numTerms()` Returns the number of terms in the document collection.

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

1.1 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 main method that does the following:

- Gets name of a folder, number of permutations to be used and an error parameter (less than one) as input. Let us use ϵ to denote the error parameter and creates and instance of `MinHash`.
- For every pair of files in the document collection, compute exact Jaccard Similarity and approximate Jaccard similarity (obtained by calling methods `exactJaccard` and `approximateJaccard` from the class `MinHash`).
- Reports the number of pairs for which exact and approximate similarities differ by more then ϵ.

1.2 MinHashSpeed

This class tests whether it is faster to estimate Jaccard Similarities using MinHash matrix whether to compute the similarities exactly. This class has a main method that

- Gets name of a folder, number of permutations to be used.
- 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.

2 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.

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

2.1 NearDuplicates

This class puts together `MinHash` and `LSH` to detect near duplicates in a document collection. This class should have a main method that gets the following information as input from the user:

- Name of the folder containing documents
- Number of Permutations to be used for MinHash
- Number of Bands to be used in locality sensitive hashing

- Similarity threshold s.
- Name of a document from the collection, *docName*.

Then this method first compute the list of documents that are more than s-similar to *docName*, by calling the method `nearDuplicates`. Note that this list may contain some `False Positives`—Documents that are less than s-similar to *docName*. Use the MinHash matrix to eliminate the `False Positive` and output the obtained list.

3 Additional Classes

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

4 Terms

Use words as terms (as opposed to shingles). You should do following pre-processing to form terms: Convert all words into lower case, remove following punctuation symbols: Period, Comman, Colon, Semi Colon, apostrophe. Remove all STOP words, any word with length less than three is a STOP word and "the" is also a STOP word.

5 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.
- Brief pseudo code for methods: `exactJaccard, minHashSig, approximateJaccard, minHashMatrix`

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 `LSH`

- To implement LSH, you need to create b (hash) tables, where b is the number bands. Though conceptually this is simple, this (may) present(s) a few implementation challenges. What are they? How did you address them?

- Your hashing algorithm: The hash function used. Note that input to the hash function is a r-tuple, where r is the number of rows in each band. Explain how you are hashing this tuple.
- Pseudocode for `nearDuplciatesOf`

Finally, run `NearDuplicates` on the files from `pa2.zip`. Run the program on at least 10 different inputs. For each input: List all the files that are returned as near duplicates. Report the number of *false positives.*

6 Suggestions

This assignment is conceptually a little more involved. I encourage you to start early and talk to me/TA for help. Before you start, please make sure you understand the notions of Jaccard Similarity, random permutations, MinHash signature, minhash matrix and locality sensitive hashing.

If M is the number of terms in the collection, then we need a random permutation from $\{1, \cdots, M\}$ to $\{1, \ldots, M\}$. There are two choices, randomly create a permutation and store the permutation. To store k permutations, you need a $k \times M$ matrix. You may also use $ax + b\%p$ as your permutation (for an appropriate choice of a prime number p). Few things to ponder: What should be choice of p? Note that $ax + b\%p$ is a one-one function from $\{1, \cdots, M\}$ to $\{1, \ldots, M\}$, not a permutation. Does this still work? Why?

7 Data

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

For each file that appears in `space.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 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 (unless you are very unlucky and your random permutations are of poor quality).

8 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.

9 What to Submit

Please submit all .java files and the report via blackboard. Your report should be in pdf format. Please note that **only one** submission is required per group.

Have Fun!