Swatie Bansal
Shubham Agrawal

COM S 535x: Programming Assignment #2

MinHash

Q. Your procedure to collect all terms of the documents and the data structure used for this.

All the punctuations and stop statements have to be removed and all other words are considered as proper terms. Because same word can appear in more than one doc, hence in order to save memory we have calcualted in how many documents a particular term appears. To achieve this, we have saved a map of string to set of integers, i.e.

HashMap<String, Set<Integer> >

Now String (key of map) signifies the terms, while in Set<Integer>, Integer signifies document numbers in which String has appeared.

To calculate this we read each doc one by one, clean the data from the document to remove stop terms and punctuation marks. Each file is mapped to one integer (simple for loop).
For each proper term appearing in document, we check if the term contains this document in its set. If term have not saved the document number in its set then we add it.

This way we use lesser memory (of course this has limitations if the documents don't share words but this is less probable) by saving set of docs against terms.

Further to achieve flexibility we take out all keys of map (i.e. terms) in an array of String, which is flexible way to access.

Q. Your procedure to assign an integer to each term.

We have saved all the terms in a map, which has terms -> set of all docs containing it. Later we took map.keys set, which is an array and then we use this for flexibility. So terms are eventually stores in array of string.

Q. The permutations used, and the process used to generate random permutations.

We created two arrays of random values and then used their addition modulo new prime number as random permutation.

Q. Brief pseudo code for methods: exactJaccard, minHashSig, approximateJaccard, minHashMatrix

exactJaccard(String filename1, String filename2): Get names of two files (in the document collection) file1 and file2 as parameters and returns the exact Jaccard Similarity of the files

- Check whether the filename1 and filename2 are present in the array list of filenames
- Calculate the length of vector T1 and T2 (representing file 1 and file 2)
- Calculate dot product of T1 and T2
- Using the formula of Jaccard Similarity: Jaccard Similarity between Da and Db, Jac (Da, Db) = Ta·Tb / [L2(Ta)]^2 + [L2(Tb)]^2 - Ta·Tb, calculate the value between filename 1 and filename2

minHashSig(String fileName): Returns MinHash the Min hash signature of the document named fileName.

- This method accepts a string, which is file name
- Create an array of int minHashSig, which is of length number of permutations given by user.
- Get the index of the file by using the file as inout to getIndex method.
- For every term calculated and for documents calculate the hash value by using array RandA and RandB (which signify randomization in the permutations.) if calculated hash value is less than known minHash then update the minHash
- Store the value of minHashSig of the document and return the minHashSig

approximateJaccard(String filename1, String filename2)

- Get the minHashSig of filename1 and filename2 for which we need to calculate the Jaccard similarity
- Calculate the number of components (count) for which minHashSig of file 1 and file 2 match.
- Output count/numofPermutations

minHashMatrix: To calcualte the minHashMatrix, we used our map (term-> set of doc).
Assuming that matrix is the minHashMatrix
Psuedo code:
1. Get all the keys of map (terms) in array words.
2. Get all the document in array files_name
3. Initialize minHashMatrix[term length][document length] to 0
4. for(i=0;i< words.length; i++){
   for (int j = 0; j < files_name.length; j++) {
    if (mapping.get((String) wordSet[i]).contains(j)) {
    // if the map has the current document in value mapped to current term then put 1 in minHashMatrix
    // else it is automatically 0
     terms[j][i] = 1;
    }
   }
 }

Swatie Bansal
Shubham Agrawal

MinHashAccuracy

Run the program with following choices of NumPermutations: 400, 600, 800 and following choices for: 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?

ϵ / NumPermutations	400	600	800
0.04	1153	148	81
0.07	7	0	0
0.09	0	0	0

MinHashSpeed

numofPermutations: 400

- Exact Jaccard Similarity time: 22421 ms
- Approximate Jaccard Similarity time: 6119 ms

numofPermutations: 600

- Exact Jaccard Similarity time: 22404 ms
- Approximate Jaccard Similarity time: 8917 ms

numofPermutations: 800

- Exact Jaccard Similarity time: 22428 ms
- Approximate Jaccard Similarity time: 10915 ms

LSH

Q. 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?

We had two options :

- 2 D array of strings
- 2 D array of sets.

Swatie Bansal
Shubham Agrawal

We used 2 D array of sets to handle collision, it allows storing 2 documents together at the same index.

Q. 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.

Hashing the tuple:
We used modular hashing.

- Hashing function: h(x) = x mod m, where m is the no of documents.
- Add all elements in the tuples and take modulo m.
- Modulo m will avoid any out of bound error.

Q. Pseudocode for nearDuplciatesOf

1. Create one MinHash object for start.
2. Create one LSH object by passing above MinHash object.
3. Calculate a list of near duplicates for given file.
4. For each document in above list, calculate approximate Jaccard Similarity using approximateJaccard function.
5. Compare above similarity with the Similarity threshold:
   5.1. If less, then it is a near duplicate document
   5.2. Else, it is false positive.

Q. 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.

NearDuplicates results -

1.

- space-0.txt.copy1
- space-0.txt.copy2
- space-0.txt.copy4
- space-0.txt.copy5
- space-0.txt.copy6

False positives - 0

2.

- space-1.txt.copy2
- space-1.txt.copy3
- space-1.txt.copy6
- space-1.txt.copy7

Swatie Bansal
Shubham Agrawal

False positives - 0

3.

space-57.txt.copy1
space-57.txt.copy2
space-57.txt.copy3
space-57.txt.copy4
space-57.txt.copy
space-57.txt.copy6

False positives - 0

4.

space-27.txt.copy1
space-27.txt.copy2
space-27.txt.copy3
space-27.txt.copy4
space-27.txt.copy5
space-27.txt.copy6
space-27.txt.copy7

False positives - 0

5.

space-48.txt.copy1
space-48.txt.copy2
space-48.txt.copy3
space-48.txt.copy4
space-48.txt.copy5
space-48.txt.copy6

False positives - 0

6.

space-119.txt.copy1
space-119.txt.copy3

False positives - 1

7.

space-301.txt.copy1
space-301.txt.copy2
space-301.txt.copy3
space-301.txt.copy4

space-301.txt.copy5
space-301.txt.copy6

False positives - 0

8.

space-859.txt.copy1
space-859.txt.copy2
space-859.txt.copy3
space-859.txt.copy4
space-859.txt.copy5
space-859.txt.copy6

False positives - 0

9.

space-635.txt.copy1
space-635.txt.copy2
space-635.txt.copy3
space-635.txt.copy4
space-635.txt.copy5
space-635.txt.copy6

False positives - 0

10.

space-775.txt.copy2
space-775.txt.copy3
space-775.txt.copy7

False positives - 1