Intelligent Data Mining - Exercise 4

Michael Debono Mrđen

14 November 2017

1 Assignment 1: Minhashing

a. Compute the minhash signature for each column if we use the following three hash functions:

- $h_1(x) = 2x + 1 \mod 6$
- $h_2(x) = 3x + 2 \mod 6$
- $h_3(x) = 5x + 2 \mod 6$

Element	S_1	S_2	S_3	S_4	h_1	h_2	h_3
0	0	1	0	1	1	2	2
1	0	1	0	0	3	5	1
2	1	0	0	1	5	2	0
3	0	0	1	0	1	5	5
4	0	0	1	1	3	2	4
5	1	0	0	0	5	5	3

b. Which of these hash functions are true permutations?

Only h_3 is a true permutation as it defines different hashes for all available elements.

c. How close are the estimated Jaccard similarities (based on the minhashes) for the six pairs of columns to the true Jaccard similarities.

Signature matrix computation:

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞
h_3	∞	∞	∞	∞

(1)

	S_1	S_2	S_3	S_4
h_1	∞	1	∞	1
h_2	∞	2	∞	2
h_3	∞	2	∞	2

(2)

$$\begin{array}{c||c|c|c|c} & S_1 & S_2 & S_3 & S_4 \\ \hline h_1 & \infty & 1 & \infty & 1 \\ h_2 & \infty & 2 & \infty & 2 \\ h_3 & \infty & 1 & \infty & 2 \end{array} \tag{3}$$

$$\begin{array}{c||c|c|c|c} & S_1 & S_2 & S_3 & S_4 \\ \hline h_1 & 5 & 1 & \infty & 1 \\ h_2 & 2 & 2 & \infty & 2 \\ h_3 & 0 & 1 & \infty & 0 \end{array} \tag{4}$$

$$\begin{array}{c||c|c|c|c} & S_1 & S_2 & S_3 & S_4 \\ \hline h_1 & 5 & 1 & 1 & 1 \\ h_2 & 2 & 2 & 5 & 2 \\ h_3 & 0 & 1 & 5 & 0 \end{array} \tag{5}$$

$$\begin{array}{c||c|c|c|c} & S_1 & S_2 & S_3 & S_4 \\ \hline h_1 & 5 & 1 & 1 & 1 \\ h_2 & 2 & 2 & 2 & 2 \\ h_3 & 0 & 1 & 4 & 0 \end{array} \tag{6}$$

$$\begin{array}{c||c|c|c|c} & S_1 & S_2 & S_3 & S_4 \\ \hline h_1 & 5 & 1 & 1 & 1 \\ h_2 & 2 & 2 & 2 & 2 \\ h_3 & 0 & 1 & 4 & 0 \end{array} \tag{7}$$

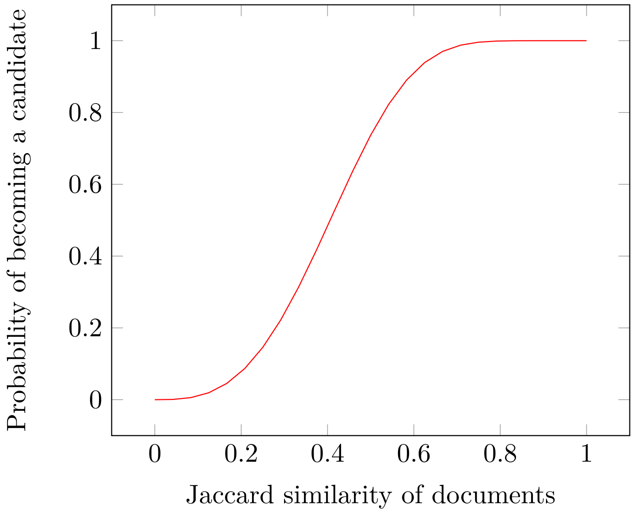
Jaccard similarities:

	Estimated	True	|Difference|
S_1, S_2	1/3	0	0.33
S_1, S_3	1/3	0	0.33
S_1, S_4	2/3	1/4	0.42
S_2, S_3	2/3	0	0.67
S_2, S_4	2/3	1/4	0.42
S_3, S_4	2/3	1/4	0.42

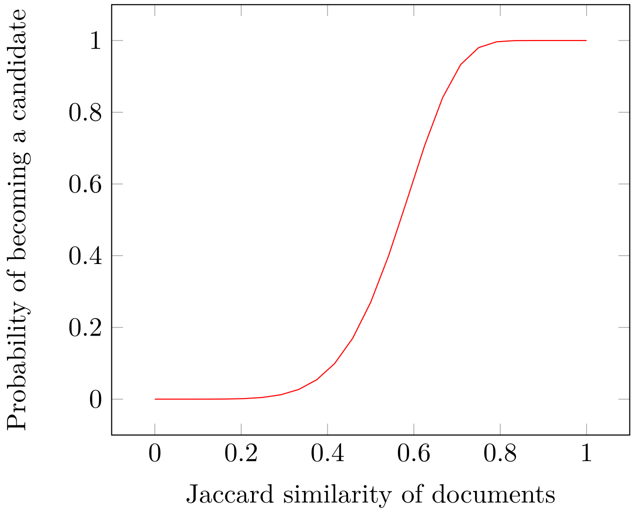
2 Assignment 2: Locality-sensitive hashing

a. Provide plots of the S-curve $1 - (1 - s^r)^b$ for the following values of r and b:

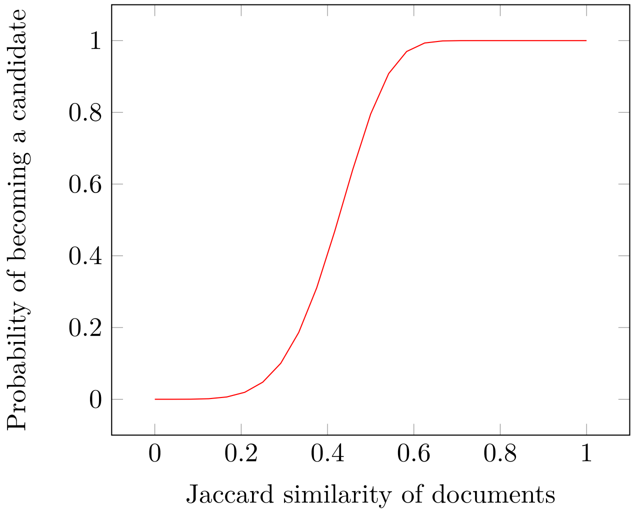
- $r = 3$ and $b = 10$



- $r = 6$ and $b = 20$



- $r = 5$ and $b = 50$



b. For each of the (r, b) pairs in (a), compute the threshold, that is, the value of s for which the value of $1 - (1 - s^r)^b$ is exactly 1/2. How does this value compare with the estimate of $(1/b)^{1/r}$ that was suggested in Section 3.4.2?

	s when $1 - (1 - s^r)^b = 1/2$	$(1/b)^{1/r}$	|Difference|
$r = 3, b = 10$	0.40609	0.46415	0.058
$r = 6, b = 20$	0.56935	0.60696	0.038
$r = 5, b = 50$	0.42439	0.45730	0.033

3 Assignment 3: Minhashing in Java

App.java:

```
package com.intelligent.data.management.Exercise4Assignment3;

import java.io.File;
import java.io.IOException;
import java.util.HashMap;

import org.apache.mahout.cf.taste.common.TasteException;
import org.apache.mahout.cf.taste.impl.common.FastIDSet;
import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import org.apache.mahout.cf.taste.model.DataModel;

public class App
```

```
{
        public static long hash1(long itemID) {
                return (itemID + 1) % 9;
        }
        public static long hash2(long itemID) {
                return (3*itemID + 1) % 9;
        }

    public static void main( String[] args ) throws IOException, TasteException
    {
        // load data
        DataModel data = new FileDataModel(new File("data/data.csv"));

        // Represent each user as a set of item IDs (note: ignore ratings).
        // >> We will use the FastIDSet in the DataModel

        // Print the characteristic matrix (see Section 3.3.1) containing all us
        // (note: sort the matrix by Item ID).

        // Choose two hash functions (similar to Figure 3.4) and compute the min
                // (similar to Section 3.3.5).

        //HashMap<Long,Long> h1s = new HashMap<Long,Long>();
        //HashMap<Long,Long> h2s = new HashMap<Long,Long>();

                // initialize user minhash signatures with max values
                HashMap<Long,Long> minh1s = new HashMap<Long,Long>();
                HashMap<Long,Long> minh2s = new HashMap<Long,Long>();
                LongPrimitiveIterator userIDsIterator = data.getUserIDs();
        while (userIDsIterator.hasNext()) {
                long userID = userIDsIterator.nextLong();
                minh1s.put(userID, Long.MAX_VALUE);
                minh2s.put(userID, Long.MAX_VALUE);
        }

        // Print characteristic matrix with hash signatures
        System.out.println("Characteristic␣matrix␣with␣hash␣signatures:");
                System.out.println();
                // Print header line
        System.out.print("␣␣␣␣␣␣␣");
        userIDsIterator = data.getUserIDs();
        while (userIDsIterator.hasNext()) {
                long userID = userIDsIterator.nextLong();
                System.out.print(String.format("␣|␣User␣%d", userID));
        }
        System.out.println("␣||␣␣h_1␣␣|␣␣h_2");
System.out.println("------------------------------------------------------------

// Loop through all items
LongPrimitiveIterator itemIDsIterator = data.getItemIDs();
while (itemIDsIterator.hasNext()) {
        long itemID = itemIDsIterator.nextLong();
        System.out.print(String.format("Item␣%d␣", itemID));
        // Compute the hashes for the item
        long h1 = hash1(itemID);
        long h2 = hash2(itemID);
        // Loop through all users
        userIDsIterator = data.getUserIDs();
        while (userIDsIterator.hasNext()) {
                long userID = userIDsIterator.nextLong();
                // Get the items for the user
                FastIDSet itemIDs = data.getItemIDsFromUser(userID);
                if (itemIDs.contains(itemID)) {
                        // If the user reviewed this item, print 1
                        System.out.print("|␣␣␣1␣␣␣␣");
                        // Store the hash value if it is less than what
                        if (h1 < minh1s.get(userID)) {
                                minh1s.put(userID, h1);
                        }
                        if (h2 < minh2s.get(userID)) {
                                minh2s.put(userID, h2);
                        }
                } else {
                        // If the user did not review this item, print 0
                        System.out.print("|␣␣␣0␣␣␣␣");
                }
        }
        //h1s.put(itemID, h1);
        // Print the hash values
        System.out.print(String.format("||␣␣␣%d␣␣␣", h1));
        //h2s.put(itemID, h2);
        System.out.print(String.format("|␣␣␣%d␣␣␣", h2));
        System.out.println();
}
        System.out.println();

        // Print minhash signature matrix for users
        System.out.println("Minhash␣signature␣matrix␣for␣users:");
        System.out.println();
        System.out.print("␣␣␣");
        // Print header line
userIDsIterator = data.getUserIDs();
```

```
while (userIDsIterator.hasNext()) {
        long userID = userIDsIterator.nextLong();
        System.out.print(String.format("␣|␣User␣%d", userID));
}
System.out.println();
// Print data for hash function 1
System.out.print("h_1");
userIDsIterator = data.getUserIDs();
while (userIDsIterator.hasNext()) {
        long userID = userIDsIterator.nextLong();
        System.out.print(String.format("␣|␣␣␣%d␣␣␣", minh1s.get(userID))
}
System.out.println();
// Print data for hash function 2
System.out.print("h_2");
userIDsIterator = data.getUserIDs();
while (userIDsIterator.hasNext()) {
        long userID = userIDsIterator.nextLong();
        System.out.print(String.format("␣|␣␣␣%d␣␣␣", minh2s.get(userID))
}
System.out.println();
System.out.println();

// Print out the resulting pairwise similarities based on the minhashes.
System.out.println("Estimated␣pairwise␣similarities:");
System.out.println();
// Loop over all users
LongPrimitiveIterator i1 = data.getUserIDs();
while (i1.hasNext()) {
        long u1 = i1.nextLong();
        // Loop again on all users for second user in pair
        LongPrimitiveIterator i2 = data.getUserIDs();
        while (i2.hasNext()) {
                long u2 = i2.nextLong();
                // Take unique pairs of users, assuming they come in ord
                if (u1 < u2) {
                        // Count similarities
                        int sim = 0;
                        if (minh1s.get(u1) == minh1s.get(u2)) {
                                sim++;
                        }
                        if (minh2s.get(u1) == minh2s.get(u2)) {
                                sim++;
                        }
                        // Divide similarity count by 2 for two hash fun
                        System.out.println(String.format("User␣%d,␣User
                        }
                }
        }
    }
}
```