Mining massive Datasets WS 2017/18

Problem Set 6

Rudolf Chrispens, Marvin Klaus, Daniela Schacherer

December 2, 2017

Exercise 01

I think of two different approaches.

1. In the first approach we count same items as if they are different. So we just ignore the property of a multiset and count every item as one, even if they are duplicates.

2. The second approach would count duplicate items with smaller weight. So for the first of many same items x the weight is 1. For every item that is identical to x the weight is either 0.5 or gets halved every time an identical item appears. I think this approach is the better of the two presented. In the case that the two parameters are just sets, every item has the weight 1.

Exercise 02

Exercise 03

Exercise 04

Given are two sets A and B. The Jaccard Similarity is zero if there are no items that are in both in A and B. When applying the minhashing algorithm it doesn't matter which permutation is used, the values for the two sets will never be the same since there are no identical values in the two sets. The similarity with the minhashing algorithm then will also be zero since every value is different.

Exercise 05

a) Jaccard similartity: $1/5 = 0.2$ (intersection/union)

b) If we choose all permutations at (120), the probability that it will produce the same minhash values for two sets is the same as the Jaccard similarity of those sets. Since the estimate of the Jaccard similarity of S_1 and S_2 is $1/5$ the fraction of 120 random permutations for those two columns that produce the same hash values is of size $120 * (1/5) = 24$.

Exercise 06

2 Other Hash function alternatives:

1) **Message-Digest algorithm 5** (MD5) (Note: counts also as not secure anymore) MD5 ist a hash function for computing a 128-bit, fixed-length message digest from an arbitrary length binary input.

2) **Fast Fourier transform algorithm** A fast Fourier transform (FFT) is an algorithm that samples a signal over a specified time and divides it into its frequency components.

R.t. A **Rainbow table** is typically stored as a Database and contains of pre calculated hashes (or encryptions) of values. (In this context password strings.) It is used to lookup "common" passwords and compare the value of the password you want to get and the rainbow table, instead of making calculations to get those specific values. This is often much faster if the password is not complex.

Salt **Salt** is a random string of characters that is different for every single user. Salt will be generated and gets merged with the user-password. After that an algorithm hashes the new string this hash gets stored into the Database.