Assignment 3: Locality Sensitive Hashing

W. Kowalczyk

wojtek@liacs.nl

10.10.2017

Introduction

The purpose of this assignment is find similar users of Netflix with help of LSH. This time, similarity between two users *u1* and *u2* is measured by the Jaccard similarity of sets of movies that they rated, while ratings themselves are irrelevant. Thus, if *Si* denotes the set of movies rated by *ui*, for i=1,2, then the similarity between *u1* is *u2* is *#intersect(S1, S2)/#union(S1, S2)*. Our objective is to find, given a set of users and movies they rated, pairs of users with similarity bigger than 0.5 - the more pairs the better. However, due to the size of the data (more than 100.000 users who rated in total 17.770 movies, each user rated at least 300 movies), instead of using brute force approach (i.e., calculating the Jaccard similarity of about 100.000*100.000/2=5.000.000.000 pairs) we will use minhashing and the LSH techniques.

Data

The data comes from the original Netflix Challenge (www.netflixprize.com). To reduce the number of users (originally: around 500.000) and to eliminate users that rated only a few movies, we selected only users who rated at least 300 and at most 3000 movies. Additionally, we recoded original user ids and movie ids by consecutive integers, so there are no 'gaps' in the data. The result was saved in a text file, *user_movie.txt*, which contains 65.225.506 records, each record consisting of two integers: *user_id* and *movie_id*, indicating that the user *user_id* rated the movie *movie_id*. For your convenience the data is also saved in numpy's binary file *user_movie.npy* as an array of integers of size (65.225.506 , 2). Both files, zipped, can be downloaded from:

https://www.dropbox.com/sh/z2ddk9384kzdlk0/AABmJbkgbKuZea1Ye7iqh3zta?dl=0

Your Task

Implement (in Python) the LSH algorithm with minhashing and apply it to the *user_movie.npy* data to find pairs of users whose similarity is at least 0.5. The output of your algorithm should be written to a text file, as a list of records in the form *u1,u2* (two integers separated by a comma), where *u1<u2* and *jsim(u1, u2)>0.5*. Additionally, as your program will rely on random number generators, set explicitly in your code the random seed to a value that will make reproduction of your results possible. The complete runtime of your algorithm should be at most 30 minutes, on a computer with 8GB RAM. The details of the submission and evaluation procedure are given below.

Submission Procedure

Submit a zipped folder with your code (without the data!) as a single file A3_your_names.zip. The folder should contain a file called main.py, which will be called (from your unzipped directory) with two arguments: a random seed (an integer) and a string that specifies the location of the *user_movie.npy* file, for example:

>> python main.py ***1234 /home/wojtek/A3/evaluation/user_movie.npy***

Your program should then load the data, set the random seed to the value provided on the command line (in our example: 1234), perform computations and dump the results to a file *results.txt* , as a list of ordered pairs: *u1, u2* (one pair per line), with *u1<u2*. Our evaluation script will check if your solution is valid: i.e., if the listed pairs really represent pairs of users with similarity bigger than 0.5, and if there are no repeating records. Set the seed of the random number generators with help of: *np.random.seed(seed='your seed')*.

Evaluation Procedure

Your program is supposed to terminate in less than 30 minutes on a machine with Intel Xeon E5-2630v3 CPUs @ 2.40Ghz, using just one core of this processor. If it doesn't terminate in 30 minutes, the process will be killed and the *results.txt* file (if exists) will be treated as the output of the run.

The baseline grade**, 6.0**, will be given if your algorithm terminates in less than 30 minutes, **producing at least 10 valid pairs of similar users**. Moreover, you can get up to **1.0** extra point for:

- the **average number of found pairs per minute,**
- the **total number of found pairs** (over 5 different runs),
- the **median** run time (over 5 different runs),
- **code readability, elegance**.

The first two criteria are "soft"; the remaining two are subject of comparing your results to results of all other submissions.

Example code

You are free to implement your solution as you like. However, you may take a look at a demo implementation of LSH and minhashing (in Matlab) . Note, that this implementation uses three tricks/shortcuts:

1) it works with a "real" 0-1 matrix that represents sets (in contrast to what is suggested in the textbook); perhaps you could use here Numpy sparse matrices?

2) minhashes are computed by randomly permuting rows of the matrix (and not by hashing them),

3) blocks of signatures are not really hashed, but aggregated with help of the unique function.

Expected results

We successfully adopted the Matlab code (from *lsh.zip*) to handle the Netflix data. Trying a few setups we ended up with one that finds about 100-200 pairs (with similarity >0.5) in 6-10 minutes on an old laptop with 8GB and an Intel P8400 processor (2.27GHz) - depending on the value of the random seed. Your Python implementation shouldn't be worse!

Hints

To make sure that your program will not exceed the 30 minutes runtime you are advice to close the result.txt file after any new pair is appended to it.

Moreover, we discovered that sometimes (depending on the value of the random seed) buckets that correspond to bands contain many candidates (false positive), making their verification very slow – just bad luck. Perhaps you can invent some work around to handle such situations?

Organization

The deadline for this assignment is **Monday, 23:59, 23.10.2017.**

The submissions should be mailed to **wojtek@liacs.nl** , in the same way as you did with A1 and A2.