

Assignment 1

Recommender Systems

Wojtek Kowalczyk
wojtek@liacs.nl

12-09-2017

Introduction

In this assignment you will work with the *MovieLens 1M* dataset which can be fetched from <http://grouplens.org/datasets/movielens/>. This set contains about 1.000.000 ratings given to about 4.000 movies by about 6.000 users. Additionally, some information is provided about movies (genre, title, production year) and users (gender, age, occupation). Your task is to implement in Python several recommendation algorithms that have been discussed during the course, and estimate their accuracy with: the Root Mean Squared Error, RMSE, and the Mean Absolute Error, MAE, with help of 5-fold cross-validation. The algorithms that you should implement are:

- Naive Approaches: the first 4 formulas from slide 17.
- Matrix factorization with Gradient Descent and Regularization as described in the paper [gravity-Tikk.pdf](#) (the beginning of section 3.1).
- Optionally, if you want a bonus: Matrix Factorization with Alternating Least Squares method, as described in the paper [ALS.pdf](#), section 3.1.

Additionally, it might be interesting (although not required!) to read an overview article on matrix factorization algorithms, [ieeecomputer.pdf](#) and a detailed presentation of a distributed version of the ALS algorithm [ALS.pdf](#).

More Details

Cross-validation

We are interested in the accuracy of recommender systems on the data that was not used in the training process. Therefore, you are required to apply the so-called 5-fold cross-validation scheme. It means that you should split your available data, at random, into 5 parts of more or less equal sizes and develop 5 models for each combination of 4 out of 5 parts. Then, each model should be applied to the part that was not used in the training process. In this way you will generate 5 different estimates of the accuracy; their average is considered to be a good estimate of the error on the future data. You may compare your results to the results reported on <http://mymedialite.net/examples/datasets.html>. We advice you to start

with applying the cross-validation scheme to the simplest recommender: the overall average score. Write a script (or a function) that splits the data (at random) into 5 folds, constructs 5 models (each one consisting of just one number: the average score) and applies these models to the training and test sets, generating predictions and calculating errors. Finally, average errors over the training sets and over the test sets to get estimates of the accuracy of your recommender, both on the training and the test sets. Repeat this process for every other recommendation algorithm. The enclosed Python program `demo_avg_rating.py` should give you an idea how your implementation could look like.

Naive Approaches

The “average rating” recommender requires no further explanation. However, when building models for “average user rating” or “average movie rating” you must take into account that during the sampling process some users or some movies might disappear from the training sets – all their ratings will enter the test set. To handle such cases, use the “global average rating” as a fall-back value.

Concerning regression models, experiment only with the “full” variant of linear regression (i.e., include the γ parameter):

$$pred = \alpha \cdot avg_{user} + \beta \cdot avg_{movie} + \gamma$$

Additionally, improve predictions by rounding values bigger than 5 to 5 and smaller than 1 to 1 (valid ratings are always between 1 and 5). Which fall-back values will you use when user or movie average rating is not available? Describe it in your report!

Thus there are 4 naive approaches: global average, user average, movie average and a linear combination of user and movie averages (with fall-back rules).

Matrix Factorization

The implementation of this algorithm is relatively straightforward. Check the blog of S. Funk, <http://sifter.org/~simon/journal/20061211.html>, and <http://www.timelydevelopment.com/demos/NetflixPrize.aspx> for an example C++ implementation of his ideas. Note that you should implement the algorithm that is described in the `gravity-Tikk.pdf` paper – their version of the algorithm is better than the one proposed by S. Funk.

Keep in mind that it may take hours to run this algorithm (a single fold should take less than one hour). Therefore, instead of running multiple runs in search for optimal parameters, run at least one experiment with the parameters reported on the MyMedialite website:

`num_factors=10, num_iter=75, regularization=0.05, learn_rate=0.005.`

The Report

In your report describe experiments that you’ve performed (including values of various parameters that you’ve used in your simulations), the accuracies achieved (both on the training and the test data) and the actual run time. To make sure that your results are reproducible, explicitly initialize random seeds to your favourite values. (There are two places where randomness plays a role: splitting data into folds and initialization of factors in the Matrix Factorization approach.)

What can you say about the required time and memory of the implemented algorithms? How quickly the required cpu-time and memory would grow with the number of movies, M , the number of users, U , and the number of ratings, R ? Informally, the “required time” is proportional to the number of steps a program has to execute and the “required memory” is proportional to the number of bytes that are used by a program. Traditionally, time and memory requirements are expressed with help of the “big O” notation, see en.wikipedia.org/wiki/Big_O_notation. For example, multiplying an $M \times N$ matrix by an $N \times K$ matrix requires $O(MNK)$ time and $O(MK)$ memory. Indeed, the result of such a multiplication is an $M \times K$ matrix, and calculating any entry of this matrix requires N multiplications and $N - 1$ additions.

Organization

As explained earlier, you should form two-person teams and work together on your solutions. The hard deadline for this assignment is **Monday, 25th September, 23:59**. Late submissions will be punished by lowering the grade by 0.1 per day of delay.

Your submission—a single zip file named **A1.zip** should contain the final report in the pdf format (at most 10 pages) and all relevant files (scripts that are necessary to reproduce your experiments). Because your work will be evaluated by your peers make sure that your report (in pdf) and your code are anonymized, i.e., contain no names, no student numbers, no email addresses.

Submit a single zip file via email to `wojtek@liacs.nl` in the following format:

E-mail Subject: **A1_Id1_Id2**

E-mail Body:

`name1, email address1`

`name2, email address2`

E-mail Attachment: **A1.zip**—a single file including your report (in the pdf format) and your code (with a short `readme.txt` file).

Id1 and **Id2** denote here the uSIS ids of the team members (like `s12345678`), while `name1`, `name2`, `email address1`, `email address2` denote their real names and email addresses. If you don’t have a uSIS ID (e.g., you are a guest or a PhD student) use your surname instead.