Assignment 1
Recommender Systems

Wojtek Kowalczyk
wojtek@liacs.nl

11-09-2018

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. To make sure that your results are reliable use 5-fold cross-validation. In other words, split your data set at random into 5 equal size pieces and use each of this piece as a test set, while training the model on the remaining 4 pieces. The average error of these five models (measured on the 5 test sets) is a reliable estimate of the accuracy of the (hypothetical) final model that is trained on the whole data set.

The algorithms that you should implement are:

- Naive Approaches: the first 4 formulas from slide 17: the global average rating, the average rating per item, the average rating per user, and an "optimal" linear combination of the 3 averages. (The global average does not depend on the specific user or item, hence it is constant - therefore we model its contribution by a single parameter γ.)
- Matrix factorization with Gradient Descent and Regularization as described in the paper `gravity-Tikk.pdf` (the beginning of 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.

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).

Thus there are 4 naive approaches: global average, user average, movie average and a linear combination of the three 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 pass through the training set could take a couple of minutes). 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. Formulate your answer to this question using the "big O" notation!

When estimating the amount of memory needed to run your algorithms, keep in mind that in "real life" the training sets might be huge! Fortunately, we don't have to load these sets into RAM–it is sufficient to process every rating one by one, perhaps several times. For example, to calculate the global mean we only need to scan the data once, summing up all the ratings (one float, Sum) and counting them (one integer N). The global mean is then S/N, so we need memory to store only 2 numbers, i.e., the required memory is $O(1)$.

Organization

As explained earlier, you should form two-person teams and work together on your assignments. The hard deadline for this assignment is **Tuesday, 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; no data!). 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 an email address that will be announced on the Blackboard later this week, in the following format:

Subject: `A1_Id1_Id2`

Body:
```
name1, email address1
name2, email address2
```

Attachment: `A1.zip`–a single file including your report (in the pdf format) and your code (with a short `readme.txt` file),

where `Id1` and `Id2` denote here the uSIS ids of the team members (student numbers, 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.