Computation and Economics - Spring 2012
Assignment #9: Collaborative Filtering

Professor Sven Seuken
Department of Informatics, University of Zurich
Out: Monday, May 14, 2012
Due: **12:15** sharp: **Thursday, May 24, 2012**
Submissions should be made to `econcs.uzh.spring12@gmail.com`.
For submission format, check description below.

[**Total: BSc 100 Points, MSc 120 Points**] This is a group-assignment to be completed by groups of **up to 2 students each**. While you are permitted to discuss your ideas with all students as much as you like, each group must write their own code and explanations. In addition, there will be a small bonus for predictors that perform well in the class competition. If you want a partner and don't have one, post to piazza as early as possible. Your submissions should be made via email to the course email address as attachments: Code in .py-files and writeup of analysis as PDF. Sending .zip-archives is also fine.

Goal

Your task in this assignment is to implement several collaborative filtering algorithms and run experiments to analyze their performance.

Introduction

- You will implement user-based and item-based collaborative filtering to predict movie ratings. The description of these algorithms can be found in the lecture notes in chapters 16.4 and 16.5, respectively.
- The test data is the MovieLens 100k data set.[1] This data set contains 100'000 ratings of 943 users on 1682 movies. It is included in the release code in the folder "ml-100k". Basically, every element in the data set is of the form $(user, movie, rating)$, where *rating* is a number between 1 and 5. Read the README file to learn more about the data format.
- To test the performance of your algorithms, we will use simple cross-validation. Cross-validation partitions the data set into two disjoint subsets, a *training set* and a *test set*. The training data is used to fit the model to the data. In our case, this means measuring the similarity between users and items, respectively, and computing the neighbourhood. After this is done, we use the test set to evualate the predictive power of the algorithm. For all

[1] Available at `http://www.grouplens.org/node/73`

items $(u, m, r)_i$ in the test set, we compute the prediction p_i and compare it to the actual rating r_i the user gave to the movie.

As precision measure we use the Root Mean Squared Error (RMSE), which is defined as

$$RMSE(\mathbf{r}, \mathbf{p}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(r_i - p_i)^2},$$

where **r** and **p** are tupels of size N containing the actual ratings and the predictions, respectively.

Setup

Source code Familiarize yourself with the provided code. You will need to implement three classes in `predictors.py`: UserBased, ItemBased, and MagicBased.

Data Sets There are different subsets available. Use the following ones for debugging, training and testing.

- **Debugging:** `small.base` and `small.test`. You can also use `ua.base` and `ua.test`.
- **Tuning:** `ua.base` and `ua.test`. Use this data set to fine-tune your algorithm (e.g. try different neighbourhood sizes etc.). Make sure you do not overfit your algorithm to this data set.
- **Testing:** `ub.base` and `ub.test`. Use this data set to actually get your final results, which you will report in your writeup.

Testing Here are some initial test commands to run. The following command gives a list of helpful command line parameters:

```
> ./rec.py --h
```

The following command runs all predictors with `ua.base` as training set and `ua.test` as test set with a neighbourhood size of 5 (the default is N=10):

```
> ./rec.py ua.base ua.test --N=5
```

The following command gives a lot of detailed output about the results of your algorithms as well as information about the data set that is being used (this is achieved by using the switch `--item_info=small.item`):

```
> ./rec.py --loglevel debug --item_info=small.item small.base small.test
```

Benchmark Algorithms In the code, there are two simple predictor algorithms already implemented against which you can test your algorithms. The first one is `AverageBased`, which always predicts the average of all ratings. The second is `AverageUserBased`, which for every user always predicts his average rating.

Tips, Comments

- Use the small test case for debugging.
- Useful data structures might be: `dict`, `set`, `list`.
- Useful commands include `itertools.groupby`, `sum(map(operator.mul, list1, list2))`, `operator.itemgetter(0)`.
- Computing similarities is quite time consuming. Try to exploit symmetries (i.e. $similarity(i, j) = similarity(j, i)$).
- Make sure to exclude all negative values of similarities when making predictions.
- When computing the similarities, you might get a division by zero error if the denominator of your coefficient is 0. In this case, make the similarity be 0.
- Please document your code well.
- If you find any bugs, please post them on NB.

Problem Set

1. [**35 Points, MSc 40 Points**] User-based Predictor

   (a) [**35 Points**] In the class `UserBased` in `predictors.py`, implement the user-based collaborative filtering algorithm. You can find the details of the algorithm in section 16.4 of the lecture notes. Use the adjusted weighted-sum aggregation method (lecture notes, Eq. 16.5) with the Pearson similarity coefficient (Eq. 16.2), and experiment with the size of the neighbourhood to pick one that works well.

   The code provides you with a skeleton of the class. In the function `precompute(.)`, you should compute all things related to the training data, i.e. similarities and neighbourhood, respectively. The function `predict(.)` then computes an actual prediction for a specific user and movie.

   (b) [**MSc 5 Points**] In addition to the adjusted weighted sum aggregation method for computing the predictions, also implement the simple averaging method (Eq. 16.3) and the weighted-sum aggregation (Eq. 16.4).

2. [**35, MSc 40 Points**] Item-based Predictor

   (a) [**35 Points**] In the class `ItemBased` in `predictors.py`, implement the item-based collaborative filtering algorithm. You can find the details of the algorithm in section 16.5 of the lecture notes. Use the adjusted cosine similarity metric, and experiment with the size of the item neighborhood.

   Again, in the function `precompute(.)`, you should compute all things related to the training data, i.e. similarities and neighbourhood, respectively. The function `predict(.)` then computes an actual prediction for a specific user and movie.

(b) [**MSc 5 Points**] In addition to the adjusted cosine similarity, also implement the cosine-based similarity.

3. [**15 Points, MSc 25 Points**] Analysis

   For all of the following tasks, explain and interpret your results, and provide data from your tests that support your statements. We are looking for concise, precise and well documented answers.

   (a) [**5 Points**] How do the UserBased and ItemBased predictors do, compared to each other? What size of neighbourhood have you found to work well?

   (b) [**5 Points**] Do the UserBased and ItemBased predictors perform better than the simple AverageBased and UserAverageBased predictor?

   (c) [**5 Points**] Is there any difference in the runtime between your implementation of the UserBased and the ItemBased predictors? If yes, can you explain them?

   (d) [**MSc 5 Points**] For the user-based algorithm, do you notice any difference in the RMSE when switching between adjusted weighted-sum aggregation, simple averaging and the weighted-sum aggregation?

   (e) [**MSc 5 Points**] For the item-based algorithm, do you notice any difference in the RMSE when switching between adjusted cosine similarity and cosine-based similarity?

4. [**15 Points**] Competition

   Implement a better predictor than the UserBased and the ItemBased ones above. We will use this predictor in a class competition. We will run your predictor against a test data set that is unknown to you, so avoid overfitting. The score for every predictor will be a weighted sum of the root mean squared error (70%) and the running time (30%).

   Some things you could try to improve the predictive power include:

   - Do some research online – there is a LOT written on collaborative filtering and related ideas. If you use other people's ideas, cite that appropriately in your writeup. You are not allowed to simply submit someone else's code.
   - Use a different similarity metric.
   - Use the additional data contained in the data set, e.g. demographics, genre etc.
   - Use a convex combination of different predictions.

   Making significant improvements is likely to require a lot of reading, coding and experimentation. This is not required, but we want to give you a chance to do so if you are interested. However, we do expect everyone to spend some time trying a few simple things and describing how they did. If they do not improve the performance, that is fine – try to understand and explain why you're getting the results your algorithm produces.

   (a) [**5 Points**] In the class `MagicBased` in `predictors.py`, implement your improved algorithm. Briefly describe how it works and what ideas you tried in order to optimize the RMSE and the running time of your algorithm. Also briefly discuss possible reasons for the success or failure of your ideas.

(b) **[10 Points]** Win the competition. The best algorithm will get 10 points, the second best 9 and so on.