Movie Recommender System Using Collaborative and Content-based Filtering

Jinglin Wang Juiting Hsu Lei Guo

New York University, NY

Motivation

- The use of recommendation systems has been so widespread today that most of the major companies, including Google, Facebook, Amazon, LinkedIn, Microsoft, and many others use recommendation systems to help improve and expand their services.

Problems

- Cold-start Problem: when the system does not have enough data for the newly come users and new products, the recommendation is nearly random
- Test-set Selection: how to divide dataset reasonably to approximate a functional testing environment
- Evaluate Function Selection: how to choose loss function that is reasonable when evaluating errors

Dataset and Methods

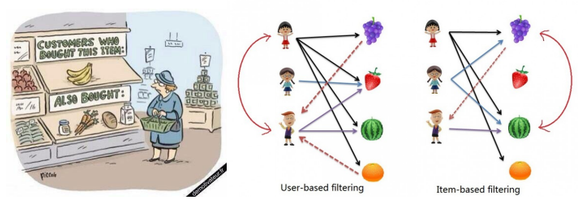
MovieLens Dataset:

- 100k dataset containing 100,000 ratings for 9,000movies by 700 users
- 1m dataset containing 1,000,000 ratings for 4,000 movies by 6,000 users.

We split 90% of the dataset into a train set and 10% into a test set while considering the chronological order of the ratings. The timestamps of the ratings of any user in the training set is always earlier than the timestamps of that user's rating in the test set. For example, if a user had 20 ratings, the earlier 18 ratings will lie in our train set, while the latter 2 ratings will be in our test set. This way, we maximally prevented ourselves from "looking into the future".

Methods:

- Baseline Model: Our baseline model simply predicts the rating by approximating the average rating for the target movie.
- Collaborative Filtering: If A likes item 1, 2, 3 and B like 2, 3, 4, then they have similar interests and A should like item 4 and B should like item 1. This algorithm is entirely based on the past behavior and not on the context.
  - Matrix Factorization
  - Slope One
  - Co-clustering
- Content-based Filtering: If you like an item then you will also like a "similar" item. It's based on similarity of the items being recommended. It generally works well when its easy to determine the context/properties of each item.

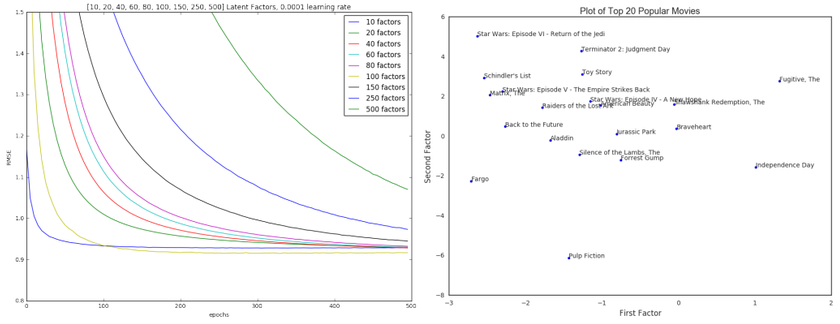


Matrix Factorization

Matrix factorization is a method that formulates the problem in a more familiar "machine learning" format by compiling the data into a matrix in which rows are users, columns are movies, and values are ratings and factorizing this matrix to obtain the latent factors w and v. In math terms, each rating is approximated by $\hat{r}_{ui} = w_u^T \cdot v_i$. The loss function on the train data is then

$$L = \frac{1}{|T|} \sum_{u,i \in T} (r_{ui} - w_u^T \cdot v_i)^2 + \lambda(\|w\|^2 + \|v\|^2)$$

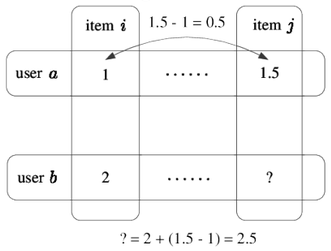
There are mainly two approaches to solving this: **Alternating Least Squares** and the good ol' **Stochastic Gradient Descent**. Shown below are the learning curves using SGD with different k (left) and the fitted models latent factors on the top 20 most popular movies (right).



Slope One

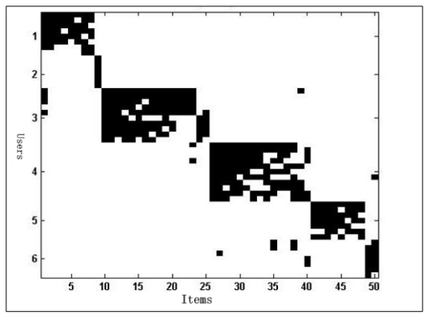
SlopeOne is the simplest form of non-trivial item-based collaborative filtering based on ratings. It has features as the following:

- Easy to implement and maintain
- Efficient at query time
- Expect little from first visitors
- Competitive accuracy



Co-clustering

The key idea is to simultaneously obtain user and item neighborhoods via coclustering and generate predictions based on the average ratings of the co-clusters (user-item neighborhoods) while taking into account the individual biases of the users and items.



Content-Based

In Content-based approach, the predicted ratings must be calculated based on the features associated with the user's preferences and compared movies. In our context, each movie could be represented with a vector model according to genres:

Action	Adventure	Animation	Children	Comedy	Crime
0	0	1	1	1	0

For a single user, the preference vector could be:

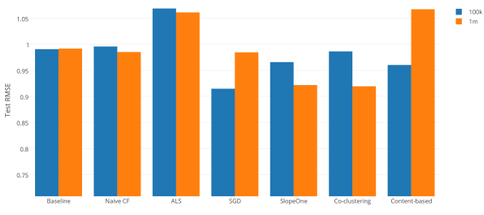
Action	Adventure	Animation	Children's	Comedy	Crime
4.2	4.0	4.0	4.2143	4.0833	4.0

For each user, by giving each genre its own weight, which depends on how often it appears in the user's rating history, the predicted rating could be computed via the equation below:

$$rating = \frac{\sum_{genre[i] \neq 1} preference[i] \times weight[i]}{\sum_{genre[i] \neq 1} weight[i]}$$

Results

Model	100k	1m
Baseline	0.9906	0.9917
Naive CF	0.9957	0.9851
Content-based	0.9602	1.0672
ALS	1.0687	1.0612
SGD	**0.9147**	0.9845
Slope One	0.9658	0.9217
Co-clustering	0.9863	**0.9193**



Some models yielded significantly improved results on both dataset. On the 100k dataset, the SGD model achieved a test RMSE of 0.9147, about 8% better than the baseline model; on the 1m dataset, the co-clustering achieved a test RMSE of 0.9193, about 8% better than the baseline model as well. The discrepancy in the performance between the two datasets might be a result of the fact that the 1m dataset actually consists of less movies than the 100k dataset.

Future Directions

- **Parallelism** When applying algorithms to big dataset like 1m MovieLens dataset, the running time of the algorithms we implemented can be significantly slowed down. For example, for the slope one algorithm, if there are m users and n movies, there'll be n^2 times of computing for each user. So for m users, the CPU computing time is mn^2. The computing time increases exponentially with the size of the dataset. One future work direction we want to work on is to take advantage of the MapReduce framework to apply parallel computing and achieve a shorter running time.
- **New Users** Our system does not take into account the prediction of new users. Most of our methods require some pre-existing user ratings in the system data. One of a possible future direction is to curate our dataset so that the system can be trained for new user rating prediction, most likely based on ratings by other users.
- **Temporal Dynamics** Another notable point is that our recommendation system is static rather than dynamic. In other words, our system is not able to factor in the possibility that users' preference might change over time. This opens up to another research area called temporal dynamics.