DS-GA 1003 - Machine Learning, Spring 2017 Movie Recommender System Using Collaborative and Content-based Filtering

Jinglin Wang, Jui-Ting Hsu, Lei Guo

May 12, 2017

1 Introduction

The use of recommendation systems has been so widespread today that many major companies, such as Facebook, Amazon, LinkedIn, Netflix, and many others use recommendation systems to help improve and expand their services. Being a hot topic during the past decade, recommendation systems today has reached a remarkable performance at predicting the users attitude towards certain products. However, there are still some problems existing in the field. First, there is the cold-start problem when the system has not enough data for the newly come users and new products, the recommendation is nearly random. The second problem is regarding the commonly used test set selection. The popular approach is to hide part of already existed data and use them as test set. The model achieves best performance on these data are used to predict those products that are really not yet reviewed. However, those real unreviewed product may be left out deliberately, because people tend to review on products they love or hate but rather those they feel lukewarm about[8]. The third problem is the evaluation function. In Netflix Prize, root mean square error is used. This evaluation function gives equal weights to all items and errors. However, those items with low review scores are hardly recommended to users and thus wrong prediction with these items are not so important as those with high scores. We're going to recommend movies to users that they might like. Our goal is to tackle the above stated problem. We'll choose reasonable test set according to time stamp and predict items that are not previously encountered by the user. Also, we'll give more weight to the movies that have extreme scores. Further more, when designing evaluation function, we're going to give more importance to errors on frequently reviewed movies and less to errors on less reviewed ones.

1.1 Data

The dataset we'll be using is the MovieLens Dataset[5] for education and development. The dataset is first released in 1998 and is now heavily downloaded thousands of times each year. It is supported by an inner rating/recommendation cycle: Users rate the items in a list of recommendations and the rates in turn affect the movies on the recommendation list. This internal cycle is enabled by Collaborative Filtering(CF) algorithm[6]. Owing to this intrinsic feature of the dataset, we tend to use CF algorithm to do the recommendation. Compared with another well-known movie dataset: the Netflix Price Dataset, which include both user ratings and other crucial information about the movies, the MovieLens Dataset emphasizes more about the users subjective opinions about the movies and the interaction between users. Also, each movie in the Movielens Dataset has been given its corresponding genres, which could greatly serve for Content-based Filtering.

We will be using two versions of the dataset: a 100k version containing 100,000 ratings for 9,000 movies by 700 users and a 1m version containing 1,000,000 ratings for 4,000 movies by 6,000 users. Note that, for example in the 100k dataset, if every user rated every movies in the dataset, there should be $9000 * 700 = 6,300,000$ ratings, but we only have 100,000 ratings. Thus, the dataset will be sparse. The dataset comes in two `csv` files: `movies.csv` and `ratings.csv`:

- In `movies.csv`, each record is a movie-id with its associated title and genres.

- In `ratings.csv`, each record is a rating by a user with user-id on a movie with movie-id of a score from 1 to 5 to the nearest 0.5.

1.2 Evaluation and Preprocessing

In order to evaluate our performance, we split 80% of the dataset into a train set, 10% into a validation set, and 10% into a test set. In order to model the scenario as close to reality as possible, we considered the timestamps of the ratings when splitting the dataset. In other words, 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 sets. To ease the processing during model fitting, we split our dataset by user groups. For example, if a user had 20 ratings, the earlier 18 ratings will lie in our train set, while the latter 1 rating will be in the validation set, and the latest rating would be in the test set. This way, we maximally prevented ourselves from "looking into the future". One thing to note is that if we're splitting our data this way, we are actually predicting the ratings for existing users rather than new users.

Since we are trying to find movies that users would enjoy, we can assess our performance by trying to predict the rating the user would give to a movie and measure the error in RMSE, or root-mean-squared error.

$$\text{RMSE} = \sqrt{\frac{1}{|U||M|}\sum_{u}^{U}\sum_{i}^{M}(\hat{r}_{ui} - r_{ui})^2}$$

where U is the set of users, M is the set of movies, $\hat{r}_{ui}$ is the predicted rating of user u on movie i, and r_{ui} is the actual rating of u on i. We note that although in an actual industrial use case, the problem is more likely to be structured as a ranking problem, we decided to use RMSE as the metric as it makes it easier to compare across the performance of different models and methods. Furthermore, RMSE is the more common metric in academic research.

2 Model

2.1 Baseline Model

A reasonable baseline algorithm for this problem is to recommend with no personalizations. In terms of accuracy evaluation, this is similar to naively predicting the ratings according to the average of the ratings. In fact, any reasonable baseline algorithm for a movie recommendation system will turn out pretty well, and will be hard to improve upon [14]. Our methods will attempt to defeat this method by employing personalization with the user ratings data provided and movie genre data. The test RMSE of the baseline model is 0.9906 on 100k and 0.9917 on 1m.

2.2 Naive Collaborative Filtering

Based on the common observation that people who agreed in evaluations of certain items in the past tends to agree in the future, collaborative filtering recommends by employing information regarding similarity between users' tastes. There are two main approaches for collaborative filtering: user-to-user and item-to-item. One problem we might come across is a sparse matrix because not every user has a rating for every movie. The cosine similarity between users u and v can be computed by

$$\text{sim}(u,v) = \frac{\mathbf{r}_u \cdot \mathbf{r}_v}{\|\mathbf{r}_u\|\|\mathbf{r}_v\|} = \frac{\sum_m^M r_{ui}r_{vi}}{\sqrt{\sum_i^M r_{ui}^2}\sqrt{\sum_i^M r_{vi}^2}}$$

where m is the number of movies and r_{ui} is the rating of user u for movie i [13]. The prediction of a user u's rating for movie i can then be computed by

$$\hat{r}_{ui} = \frac{\sum_{u'}^U \text{sim}(u,u')r_{u'i}}{\sum_{u'}^U |\text{sim}(u,u')|}$$

where U is the set of all the users other than u. The test error of this method is 0.9957 on 100k and 0.9851 on 1m, compared to baseline's 0.9906 and 0.9917. This method yielded slightly worse than baseline performance on 100k and slightly better on 1m. This is probably because this method demands more data than the baseline model does.

2.3 Content-based Filtering

The main idea of Content-based approach is very similar to that of CF: the system learns to recommend items that are similar to the ones that the user liked in the past. In our context, the difference is that the predicted ratings must be calculated based on the features associated with the compared movies[11]. For example, if we know that a user tends to positively rate movies which belong to the comedy genre, then the system should most likely recommend other movies from this genre.

2.3.1 Vector Space Model

The first challenge we are faced with is to properly represent and compare the items and the user profile. In our setting, we choose to use the "Vector Space Model". Since each movie has labels of genres it belongs to, we can represent each movie by a simple "genres vector", in which if this movie belongs to a corresponding genre, for example, "comedy", then the value corresponding to "comedy" is 1, otherwise it will be zero[2], like one-hot encoding. Please see the table below an example of "Toy Story" in the movie dataset. Note that there are 18 genres in total, but we only listed 6 of them here.

Movie	Action	Adventure	Animation	Children	Comedy	Crime	Documentary
Toy Story	0	0	1	1	1	0	0

After we derived the vector space representation of movies, the vector representation of users' preferences could be extracted from the users' rating history. For each movie a user has rated, we could use the rating as "weight" and multiply the movie vector by the weight, giving us the "rating vector". We could then take the mean of all these vectors to generate the initial vector representations for the user's preferences, or the "preference vector". The table below lists a user's rating on "Toy Story" and the corresponding rating vector.

Rating	Movie	Action	Adventure	Animation	Children	Comedy	Crime	Documentary
4.0	Toy Story	0	0	4.0	4.0	4.0	0	0

2.3.2 Generating Predictions

The most intuitive way to generate the predictions is to use cosine similarity. However, simply using cosine similarity would not generate ratings in the range of 0-5, thus making the computation of RMSE difficult. To make our model get closer to the reality, we came up with two methods:

- If a user tends to watch certain genres of movies, these genres should have larger weight than the others. So we could also derive a weight vector. This would also make the final predictions in the range of 0-5.
- Since the history records of some users are not very rich, even with the method above, ratings of some genres will not be valid for some users. In these circumstances, we choose to use the global average as the rating.

Suppose that a user has the following non-weighted "preference vector":

UserId	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary
1	4.2	4.0	4.0	4.2142	4.0833	4.0	0.0

and the following genre weights:

UserId	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary
1	0.1063	0.0851	0.2553	0.2979	0.2553	0.0425	0.0

To predict the user's rating for "Toy Story", rather than simply computing the inner product between user's preference vector and "Toy Story"'s genre vector, we should also consider the weight vector in our computation:

$$\hat{r}_{ui} = \frac{4.0 \times 0.2553 + 4.2142 \times 0.2979 + 4.0833 \times 0.2553}{0.2553 + 0.2979 + 0.2553}$$

In general, the predicted rating could be computed via the equation below:

$$\hat{r}_{ui} = \frac{\sum_{\text{genre}_j \neq 1} \text{preference}_j \times \text{weight}_j}{\sum_{\text{genre}_j \neq 1} \text{weight}_j}$$

Note that, for example, if a movie belongs to the genre "Documentary", the predicted rating would be the global average of all users regarding to "Documentary". This could definitely be improved and we talk about it in future work.

2.3.3 Results and Discussions

- For the 1m dataset, our model gives a RMSE for predictions on test dataset of 1.0673. Considering that a few predictions are based on global average we have mentioned above, and also compared to other approachs we have implemented, we think the RMSE is rather good.
- For the 100k dataset, our model gives a RMSE for predictions on test dataset of 0.9602. The reason why the smaller dataset gives us a better performance is that even though total number of ratings in this dataset is much smaller than which of the 1M dataset, most users in this dataset has rated movies in most of the 18 genres, thus the circumstances where we are forced to use the global average is much more rare.

2.4 Matrix Factorization

Matrix factorization [12] is another method that structures the problem in a more familiar "machine learning" format. By compiling the data into a big matrix in which rows are users, columns are movies, and values are ratings, matrix factorization achieves to solve the prediction problem by factorizing this matrix to obtain the latent factors. Matrix factorization assumes that:

- Each user can be described by k features. A feature i might represent a user's preference towards comedy.
- Each item can be described by k features. A feature i might represent how much a movie fits into the comedy genre (note how this is analogous to the feature in the user factors).
- A user's rating on a movie can be approximated by taking the dot product of these two vectors.

In other words, formulating the same way as before, the prediction of user u's rating for movie i can be obtained by

$$\hat{r}_{ui} = w_u^T \cdot v_i = \sum_{i=1}^{k} w_{uk} v_{ki}$$

From this, we can simply formulate our loss function using the square-loss evaluation metric as

$$L = \frac{1}{|T|} \sum_{u,i \in T} (r_{ui} - \hat{r}_{ui})^2 = \frac{1}{|T|} \sum_{u,i \in T} (r_{ui} - w_u^T \cdot v_i)^2$$

Adding in regularization terms, this becomes

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

This equation can then be solved in two approaches: alternating least squares and stochastic gradient descent[3].

2.4.1 Alternating Least Squares

Contrary to ordinary least squares, alternating least squares (ALS) provides a solution to solving the loss function over *two* vectors, w and v. ALS achieves this by holding one vector constant while setting the derivative with respect to the other vector to zero. For example, the derivation of the user vector w, while holding v constant, is as follows:

$$\begin{aligned}
\frac{\partial L}{\partial w_u} &= -2\sum_{i}^{M}\left(r_{ui} - w_u^T \cdot v_i\right) v_i^T + 2\lambda w_u^T \\
0 &= \sum_{i}^{M}\left(r_{ui} - w_u^T \cdot v_i\right) v_i^T + \lambda w_u^T \\
&= -(r_u - w_u^T \cdot V^T)V + \lambda w_u^T \\
w_u^T &= r_u V (V^T V + \lambda I)^{-1}
\end{aligned}$$

where V is a $M \times k$ matrix representing each movie using the item latent factors. Similarly, the solution for the item vector v can be solved as

$$v_i^T = r_i U (U^T U + \lambda I)^{-1}$$

where U is a $|U| \times k$ matrix representing each user using the users latent factors.

2.4.2 Stochastic Gradient Descent

Another solution to this problem is stochastic gradient descent. With SGD, we still take the gradient of each variable (weight vector), and update the weights using the gradient of the loss one sample at a time. It is also useful to include bias terms b_u and b_i for better results. The weight updates for each of the variables are as follows:

$$\begin{aligned}
b_u &\leftarrow b_u - \eta\left(\hat{r}_{ui} - r_{ui} + \lambda b_u\right) \\
b_i &\leftarrow b_i - \eta\left(\hat{r}_{ui} - r_{ui} + \lambda b_i\right) \\
w_u &\leftarrow w_u - \eta\left((\hat{r}_{ui} - r_{ui})v_i + \lambda w_u\right) \\
v_i &\leftarrow v_i - \eta\left((\hat{r}_{ui} - r_{ui})w_u + \lambda v_i\right)
\end{aligned}$$

With this formulation, we have three hyper-parameters to tune for: regularization parameter λ, dimension of latent factor k, and learning rate η. For learning rate, we just took a small value (0.0005) and run for more epochs while decaying for stability. A larger constant learning rate will result in something like Figure 1.

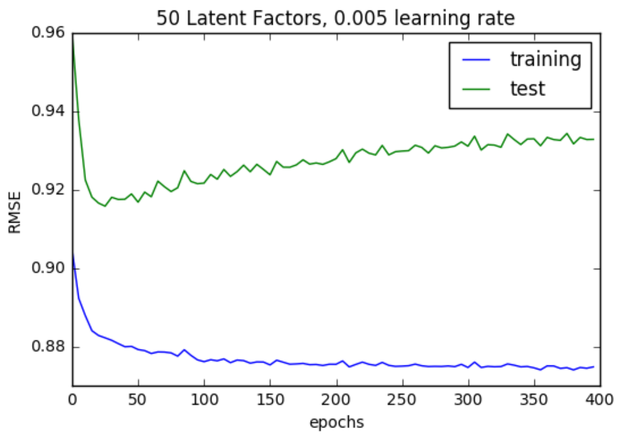

Figure 1: Undesirable earning curve when using a larger constant step size like 0.005.

Below, we show the training error and validation error we obtained when tuning the parameters k and λ. Note that in the plots, training curves were left out for better readability and comparison between different parameters.

k	Train	Validation
10	0.8840	0.9277
20	0.8959	0.9298
40	0.8914	0.9280
60	0.8890	0.9290
80	0.8869	0.9321
100	**0.8735**	**0.9134**
150	0.8815	0.9458
250	0.8754	0.9725
500	0.8548	1.0686

λ	Train	Validation
0.01	0.5281	1.2399
0.1	0.7189	1.0098
1	**0.8754**	**0.9126**
10	0.8994	0.9392
100	0.8872	0.9368

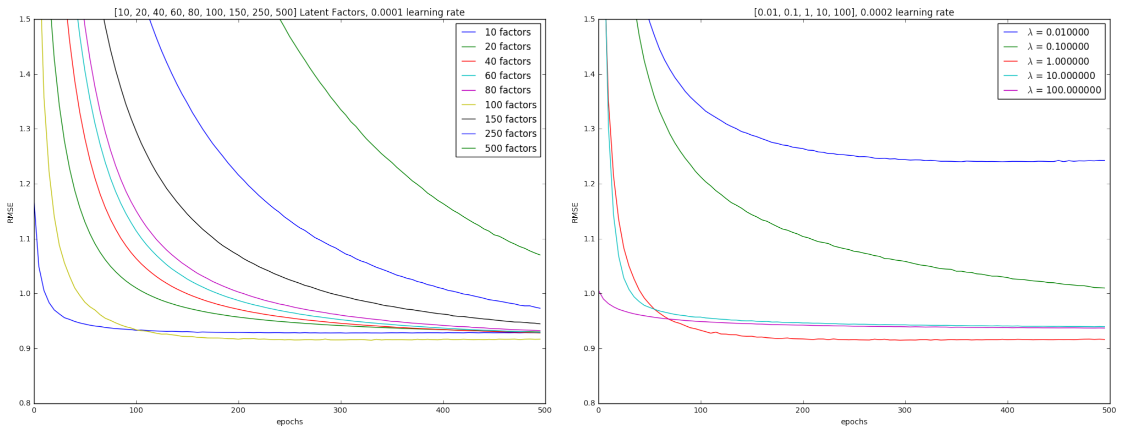

Figure 2: Comparison between performance using different k and λ.

As a “bonus” of matrix factorization-based methods, the item vectors can be extracted from the trained model and used to represent items. In this case, our item matrix will consist of movies and their “genre” scores. Below, we show a plot of the most popular 20 movies on the dataset using the first and second factors of the item vectors. People who are familiar with the movies may be able to extract meaning from these factors.

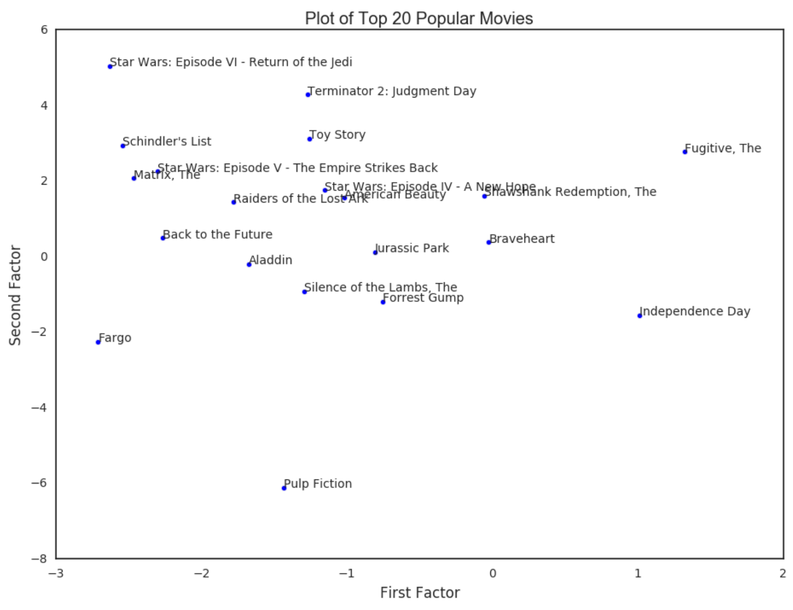

Figure 3: Plot of top 20 movies using the first and second factor of the item vector.

2.5 Slope One

Compared with model-based recommender algorithms such as singular value decomposition(SVD) and Paragon Learning Style Inventory (PLSI), Slope One algorithm is easier to implement and faster when generating prediction. The trained model can be updated with new incoming data while model-based algorithms are mostly deployed in static settings. Also, rating prediction of Slope One can be made based on only a few ratings from existing users with satisfactory accuracy.[1]

The Slope One algorithm predict an unknown rating of a particular user on an item based on rating information of the particular user's rating of other items and of other users' ratings on the same item.

To implement this algorithm, given a training set X, and two items j and i with ratings u_j and u_i with respect to some user u in the data set $(u \in S_{j,i}(X))$. We need to compute the average deviation of item i and j, which is:

$$dev_{j,i} = \sum_{u \in S_{j,i}(X)} \frac{u_j - u_i}{c(S_{j,i}(X))}$$

where $c(S_{j,i}(X) = |S_{j,i}(X)|$, the cardinality of the set. Given the above deviation, we may predict unknown ratings with the average of all such predictions

$$P(u)_j = \frac{1}{c_{R_j}} \sum_{i \in R_j} (dev_{j,i} + u_i)$$

where $R_j = \{i | i \in S(u), i \neq j, c(S_{j,i}(X)) > 0\}$ is the set of all relevant items.

However , this implementation ignores the effect of the number of ratings. When two items are predicted with different number of ratings by other users, the prediction of the item with larger number of ratings is intuitively more confident than the prediction of the other item with less ratings. To solve this problem, weighted Slope One algorithm is used to add weight to the item with more ratings. In this setting we still first compute the average deviation $dev_{j,i}$ of item i and j as stated above. Then, we can predict unknown ratings with

$$P(u)_j = \frac{\sum_{i \in S(u)} (dev_{j,i}) c_{j,i}}{\sum_{i \in S(u)} c_{j,i}}$$

where $c_{j,i}$ is the cardinality of the set of all users rating both movies[1].

The RMSE of Slope One algorithm is 0.9658 on the 100k dataset and 0.9217 on the 1m, dataset which is very good considering the simpleness of the algorithm.

2.6 Co-clustering

The above techniques can only be deployed in a static settings because their training components are computationally expensive. In real-life scenarios, we need to do real-time collaborative filtering which can support dynamic stream of new users, movies and ratings.[4] So here we also applied co-clustering to our datasets to get a sense of dynamic collaborative filtering. It is highly convenient when we add new data to the existing dataset.

To implement this algorithm, we can first consider a standard approach of matrix decomposition. Consider a user set $U = \{u_i\}_{i=1}^{m}$ and a movie set $P = \{p_j\}_{j=1}^{n}$. And an $m \times n$ matrix A where A_{ij} is the rating movie p_j from user u_i. Let W be a $m \times n$ matrix which represents the confidence of ratings in A. If the confidence information is not assigned, $W_{ij} = 1$ when there is a rating or $W_{ij} = 0$ otherwise. In other approaches like SVD, a small change in rating matrix A can lead to a dramatic change in optimization parameters of matrix decomposition. In this case, we can use simultaneous clustering of matrix A. Let $\rho : \{1, \dots, m\} \to \{1, ..., k\}$ and $\gamma : \{1, ..., n\} \to \{1, ..., l\}$ be the user and item clusters where k and l are number of user and item clusters as illustrated below. In this setting, number of parameters is kl which is not enough for a good matrix approximation. Thus biases are included.

$$\hat{A}_{ij} = A_{gh}^{COC} + (A_i^R - A_g^{RC}) + (A_j^C - A_h^C C)$$

where $g = \rho(i), h = \gamma(j)$. And the second term $(A_i^R - A_g^{RC})$ is (average ratings of u_i - average ratings of the user cluster) and the last term $(A_j^C - A_h^C C)$ is (average ratings of the movie p_j - average ratings of the movie cluster). The prediction of unknown ratings are made by finding the optimal user and item clustering that minimize the weighted approximation error of $\hat{A}$ and the original matrix A,[4] which is

$$\min_{(\rho,\gamma)} \sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij}(A_{ij} - \hat{A}_{ij})^2$$

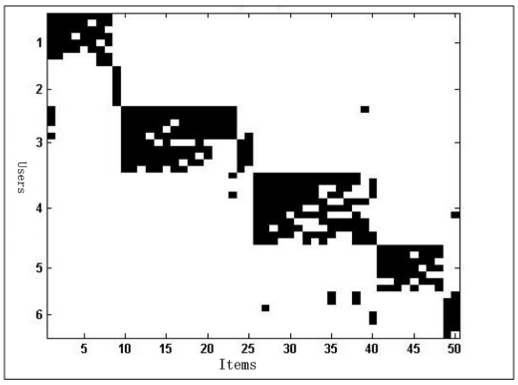

Figure 4: Co-clustering of users and items.

Applying the co-clustering algorithm to the 100k and the 1m dataset, the parameters are chosen and corresponding results are as the following, $n_c ltr_u$ is the number of user clusters and $n_c ltr_i$ is the number of movie clusters, epoch is set to be 20:

n_{user}	n_{item}	100k	1m
1	1	**0.9863**	0.9447
1	10	0.9863	0.9447
1	100	0.9863	0.9447
10	1	0.9863	0.9447
10	10	1.0142	**0.9193**
10	100	1.0495	0.9233
100	1	0.9863	0.9447
100	10	1.0276	0.9211
100	100	1.1144	0.9211

3 Results

The three problems we faced when constructing a recommender system are cold-start problem, test set selection, and choice of loss function. The cold-start problem is solved by asking new incoming users to randomly rate a few movies before getting into the recommender system. The second problem is solved by dividing the dataset according to user and timestamps. We first group the dataset by users and within each user group we divide the movies he/she watched according to timestamps. This insures the users in the test set always have rating records in the training set and the timestamps of ratings records in the training set are always before those in the test set. This setting also avoid cold-start problem when we're doing prediction on test set. For third problem, the loss function we chose is the standard RMSE. However, in the co-clustering algorithm, it includes a weight matrix in the loss function. Since the items with low ratings are hardly recommended to the users, so prediction error on them are not as important as the error on the highly rated items. So we add more weight to the items with high ratings in the rating matrix to give a more accurate result.

The best models of our result achieved test RMSE that are almost on-par with benchmark results by Surprise[7] and ACM RecSys[10].

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**
Surpriselib (best)	0.9200	0.8738

The two datasets we are using have major difference in the number of movies and the number of users. 100k dataset contains 100,000 ratings for 9000 movies by 700 users. 1m dataset contains 1,00,000 ratings for 4,000 movies by 6000 users. SGD yielded a 9% improvement from the baseline on the 100k dataset while only performing slightly better than the baseline model on the 1m dataset. The reason to this might be due to the different structure of the dataset or the difference in parameter settings. Specifically, the 1m dataset, although containing more ratings, consists of only 4000 movies compared to the 9000 movies of the 100k dataset. However, to achieve a better comparison, we maintained the parameter from the 100k dataset. A larger dataset might also require more epochs. ALS did not perform well on either dataset most likely due to the lack of data. However, ALS can be easily parallelized in the case of large-scale recommendation system computing.

Prediction results using slope one is not as good as SGD for the 100k dataset but is significantly improved when applying to 1m dataset when compared to other algorithms and the baseline. The reason for that is the slope one make prediction based on the correlation between existing ratings of the item to be predicted and all other items. There's not enough ratings information as in 100k dataset, so the prediction is not accurate. In contrast, the 1m dataset provide a lot more rating information to give a more reasonable rating correlation. Overall, slope one is a simple yet powerful algorithm.

As we can see, the Content-based approach works better when applied to the 100k dataset than the 1m dataset when compared to their baselines respectively. This is very likely due to the fact that in the 1m dataset, many genres have not enough ratings to make a convincing predict while we have to use global average of the ratings in a genre when a user never watch any movie in this particular genre before. As a contrast, there are more movies in the 100k dataset and so that they cover most of 19 genres. And most users have rated movies in all 19 genres, even though the total number of ratings is much smaller when compared to that of the 1m dataset.

As for co-clustering algorithm, it performs better when applied to 1m dataset, actually the best among all algorithms we applied. This may because it has the weight matrix in the loss function which improve the performance. Also, we tried different number of user and item clusters to get a sense of how these parameters are related to the accuracy. We can see from table above in the co-clustering section that the accuracy reach optimum at certain k and then get worse with increasing k.

Overall, for movielens dataset, because of its intrinsic feature (inner rating/recommendation cycle) as described in the data section, collaborative filtering generally performs better than content-based filtering. And when we have a dataset with large movie/user ratio, we tend to use SGD approach. And if a dataset has small movie/user ratio, we tend to use co-clustering approach.

4 Future Works

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.

In Content-based Filtering, when the rating of a user's preference on certain genres is unconvincing, we chose to use the global average instead. However, it is very likely that a user who enjoys "Animation" would enjoy "Comedy" as well. In sum, if we could find the relationships among users' ratings on different genres, then our predictions on genres where users' preference scores are unconvincing could be more accurate.

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[9].

5 Acknowledgement

We want to thank Dr. Vitaly Kuznetsov for his insightful suggestions throughout the whole project and Professor David Rosenberg and Dr. Brett Bernstein for giving us such a great course. Also, we want to thank movielens dataset and all open source packages we have used.

References

[1] Daniel Lemire, A. M. Slope one predictors for online rating-based collaborative filtering.

[2] Das, S. Beginners guide to learn about content based recommender engines. https://www.analyticsvidhya.com/blog/2015/08/beginners-guide-learn-content-based-recommender-systems/.

[3] Funk, S. Netflix update: Try this at home. http://sifter.org/~simon/journal/20061211.html.

[4] George, T. A scalable collaborative filtering framework based on co-clustering. 625–628.

[5] GroupLens. Movie Lens. https://grouplens.org/datasets/movielens/.

[6] Harper, F. M., and Konstan, J. A. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems Vol. 5*, No. 4.

[7] Hug, N. Surprise: A python scikit for recommender systems. http://surpriselib.com/.

[8] Jannach, D., Resnick, P., Tuzhilin, A., and Zanker, M. Recommender systems beyond matrix completion. *Communications of The ACM Vol. 59*, No. 11.

[9] Koren, Y. Collaborative filtering with temporal dynamics. *ACM SIGKDD international conference on Knowledge discovery and data mining* (2009).

[10] RecSysWiki. Movielens 100k benchmark results. http://www.recsyswiki.com/wiki/MovieLens_100k_benchmark_results.

[11] Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. Recommender systems handbook.

[12] Rosenthal, E. Explicit matrix factorization: Als, sgd, and all that jazz. https://blog.insightdatascience.com/explicit-matrix-factorization-als-sgd-and-all-that-jazz-b00e4d9b21ea.

[13] Rosenthal, E. Intro to recommender systems: Collaborative filtering. http://blog.ethanrosenthal.com/2015/11/02/intro-to-collaborative-filtering.

[14] Wen, Z. Recommendation system based on collaborative filtering.