MOVIE RATING ESTIMATION AND RECOMMENDATION

Lichao Zhang

1. Introduction

Recommender systems provide users with personalized suggestions for products or services. They are becoming more and more important in the success of electronic commerce, and being utilized in various applications such as Amazon, YouTube and Google news. Generally speaking, a recommendation system builds up items' profiles, and users' profiles based on their previous behavior recorded. Then it makes a prediction on the rating given by certain user on certain item which he/she has not yet evaluated. Based on the prediction, the system makes recommendations. Various techniques for recommendation generation have been proposed and successfully deployed in commercial environments, among which collaborative filtering (CF) and content-based methods are most commonly used .

Movie is now an indispensable entertainment in human life. Most video websites such as YouTube and Hulu and a number of social networks allow users rate on videos/movies. In this project, we build a movie rating prediction system based on several selected training sets, which estimates the movie ratings from each user. According to this result, we are able to make personalized movie recommendations for every user, which would more likely satisfy users' expectation and improve user experience.

In this project I built a movie recommender using collaborative filtering with Spark's Alternating Least Saqures implementation.

2. RELATED WORK

The roots of recommender systems can be traced back to the extensive work in cognitive science [1], approximation theory [2], information retrieval [3], forecasting theories [4], and also have links to management science [5] and to consumer choice modeling in marketing [6], recommender systems emerged as an independent research area in the mid-1990s when researchers started focusing on recommendation problems that explicitly rely on the ratings structure. In its most common formulation, the recommendation problem is reduced to the problem of estimating ratings for the items that have not been seen by a user. Intuitively, this estimation is usually based on the ratings given by this user to other items and on some other information that will be formally described below. Once we can estimate ratings for the yet unrated items, we can recommend to the user the item(s) with the highest estimated rating(s).

In recommender systems, the utility of an item is usually represented by a rating, which indicates how a particular user liked a particular item, e.g., John Doe gave the movie "Harry Potter" the rating of 7 (out of 10). However, as indicated earlier, in general, utility can be an arbitrary function, including a profit function. Depending on the application, utility u can either be specified by the user, as is often done for the user-defined ratings, or is computed by the application, as can be the case for a profit-based utility function.

Extrapolations from known to unknown ratings are usually done by 1) specifying heuristics that define the utility function and empirically validating its performance and 2) estimating the utility function that optimizes certain performance criterion, such as the mean square error.
The new ratings of the not-yet-rated items can be estimated in many different ways using methods from machine learning, approximation theory, and various heuristics. Recommender systems are usually classified according to their approach to rating estimation and, in the next section, we will present such a classification that was proposed in the literature and will provide a survey of different types of recommender systems. The commonly accepted formulation of the recommendation problem was first stated in [7], [8], [9] and this problem has been studied

extensively since then. Moreover, recommender systems are usually classified into the following categories, based on how recommendations are made [10]:

. Content-based recommendations: The user will be recommended items similar to the ones the user preferred in the past;

. Collaborative recommendations: The user will be recommended items that people with similar tastes and preferences liked in the past;

. Hybrid approaches: These methods combine collaborative and content-based methods.

In addition to recommender systems that predict the absolute values of ratings that individual users would give to the yet unseen items (as discussed above), there has been work done on preference-based filtering, i.e., predicting the relative preferences of users [11], [12], [13], [14]. For example, in a movie recommendation application, preference based filtering techniques would focus on predicting the correct relative order of the movies, rather than their individual ratings. However, this paper focuses primarily on rating-based recommenders since it constitutes the most popular approach to recommender systems.

3. Data Sets

I use two datasets from movieLens which are “ml-latest-small” and “ml-1m”.First,I use “ml-latest-small ”to build the basic recommender model and select optimal parameters.Then I use the ml-1m to build the final recommender model using the optimal parameters.I tried to use the bigger dataset to train my model but they called”ERROR:out of memory”.And I notice that the dataset s’ frame are the same.I guess my model will work out if I have more memory.

ml-latest-small:This dataset (ml-latest-small) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 105339 ratings and 6138 tag applications across 10329 movies. These data were created by 668 users between April 03, 1996 and January 09, 2016. This dataset was generated on January 11, 2016.Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.The data are contained in four files, links.csv, movies.csv, ratings.csv and tags.csv. All ratings are contained in the file ratings.csv. Each line of this file after the header row represents one rating of one movie by one user, and has the following format:
userId,movieId,rating,timestamp

ml-1m:These dataset contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000 provided.The data frame of “rating”and “movies” is same as ml-latest-small.The data are contained in four files, links.dat, movies.dat, ratings.dat and tags.dat.I just change them to “csv” so that they can be loaded by spark.

More details about the contents and use of all these files see :http://files.grouplens.org/datasets/movielens/ml-latest-small-README.html and http://files.grouplens.org/datasets/movielens/ml-1m-README.txt

4. Proposed Solution

The way I found to build a movie recommender is by using collaborative filtering with Spark's Alternating Least Saqures implementation.Typically, the workflow of a collaborative filtering system is:
1.A user expresses his or her preferences by rating items (e.g. movies in this project) of the system. These ratings can be viewed as an approximate representation of the user's interest in the corresponding domain.
2.The system matches this user’s ratings against other users’ and finds the people with most "similar" tastes.

3.With similar users, the system recommends items that the similar users have rated highly but not yet being rated by this user (presumably the absence of rating is often considered as the unfamiliarity of an item).
Collaborative filtering aims to fill in the missing entries of a user-item association matrix. MLlib currently supports model-based collaborative filtering, in which users and products are described by a small set of latent factors that can be used to predict missing entries. By using the alternating least squares (ALS) algorithm to learn these latent factors.

4.1 Matrix Factorization Model

Matrix factorization models map both users and items to a joint latent factor space of dimensionality *f*, such that user-item interactions are modeled as inner products in that space. Accordingly, each item *i* is associated with a vector q_i $\mathbf{R}^f$, and each user *u* is associated with a vector p_u $\mathbf{R}^f$. For a given item *i*, the elements of q_i measure the extent to which the item possesses those factors, positive or negative. For a given user *u*, the elements of p_u measure the extent of interest the user has in items that are high on the corresponding factors, again, positive or negative. The resulting dot product, q ^{T} p , captures the interaction between user *u* and item *i*—the user's overall interest in the item's characteristics. This approximates user *u*'s rating of item *i*, which is denoted by r_{ui}, leading to the estimate[15]:

$$\hat{r}_{ui} = q_i^T p_u.$$

To learn the factor vectors (p_u and q_i), the system minimizes the regularized squared error on the set of known ratings[15]:

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2)$$

Here, is the set of the (*u*,*i*) pairs for which r_{ui} is known (the training set).

4.2 Learning Algorithm- Alternative Least Squares

ALS techniques rotate between fixing the *q* 's and fixing the *p* 's. When all *p* 's are fixed, the system recomputes the q_i's by solving a least-squares problem, and vice versa. This ensures that each step decreases Equation of the RMSE until convergence.

I think ALS is favorable in at least two cases. The first is when the system can use parallelization. In ALS, the system computes each q_i independently of the other item factors and computes each *p* independently of the *u* other user factors. This gives rise to potentially massive parallelization of the algorithm.[9] The second case is for systems centered on implicit data. Because the training set cannot be considered sparse, looping over each single training case—as gradient descent does—would not be practical. ALS can efficiently handle such cases.

4.3 Basic Pineline of my project

1)Load the data :the ml-latest-small dataset for original model building
2)Preprocessing :check if there is any missing values or wrong format.If so,then delete them.
3)Using ALS to build a original recommender model
4)Using cross validation to select optimal parameters of ALS model
5)Train the model using the complete data so that to build our final model
6) Add a new user to emulate the scenario that the model would give him/she recommending movies were seen by more than 25 people given his/her formal ratings.
The criteria that I used for selecting optimal parameters is by comparing RMSE.More details are in my codes.
The function is :

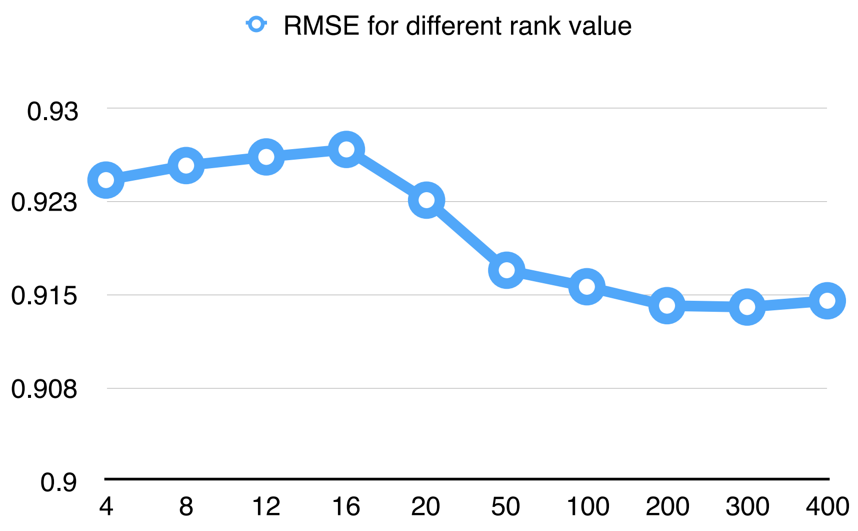
$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|^2 + \| p_u \|^2)$$

5. Experiments and Results

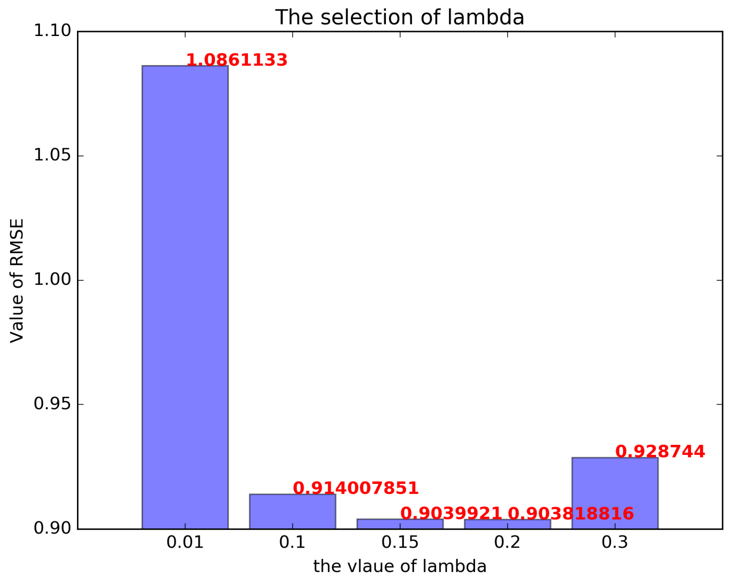
The implementation in MLlib.ALS has the following parameters:

rank is the number of latent factors in the model.
iterations is the number of iterations to run.
lambda specifies the regularization parameter in ALS.

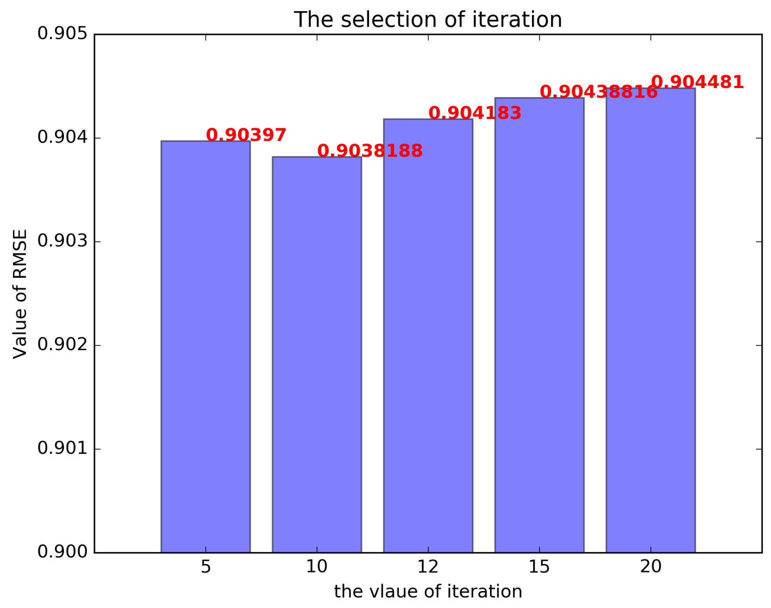
First,I tested the value of RMSE for different rank parameters with lambda = 0.2, iteration =10. We can see that the least RMSE value is 0.914020503632 for rank = 300.



By comparing different values of rank,I found out that there’s a huge drop between [16,50].So,I computed the value of RMSE when lambda = 0.01, 0.1 ,0.15, 0.2, 0.3 with rank = 4, 8, 12, 16 ,20 , 50.
The least value of RMSE is 0.914007851329 when rank= 8,lambda = 0.2 .The RMSE showed below the least value with rank =[4,20].



For selecting the value of iteration,I set values for rank =8,lambda = 0.2,then test iteration = 5,10,12,15 ,20.



At last,the best RMSE I get for ml-latest-small is 0.903818816791.
The least RMSE for ml-1m using these parameters is 0.9127177318.

6. Conclusion and Discussion

I think the major part of job for this project is to understand what is recommender system , ALS algorithm and build a movie recommender using spark.I think the problem I didn't expect is setting spark environment is pretty messy.And spark's API documents are not really well

Algorithms Compare for MovieLens Datasets

Algorithm	RMSE
KNN	0.95
Stochastic Gradient Descent	0.945
SVD	0.94
SVD++	0.928
Asymmetric SVD	0.925
Co-clustering by BVD(Block Value Decomposition)	0.917
ALS	0.912

explained.Besides,the code I include for this project is using ml-1m.In fact,it can be used for ml-latest(and I did).See more details in read_me.txt.
 Compared to other algorithm (using RMSE criteria) :

References

[1] E. Rich, “User Modeling via Stereotypes,” Cognitive Science, vol. 3, no. 4, pp. 329-354, 1979.
[2] M.J.D. Powell, Approximation Theory and Methods. Cambridge Univ. Press, 1981.
[3] G. Salton, Automatic Text Processing. Addison-Wesley, 1989.
[4] J.S. Armstrong, Principles of Forecasting—A Handbook for Researchers and Practitioners. Kluwer Academic, 2001.
[5] B.P.S. Murthi and S. Sarkar, “The Role of the Management Sciences in Research on Personalization,” Management Science, vol. 49, no. 10, pp. 1344-1362, 2003.
[6] G.L. Lilien, P. Kotler, and K.S. Moorthy, Marketing Models. Prentice Hall, 1992.
[7] W. Hill, L. Stead, M. Rosenstein, and G. Furnas, “Recommending and Evaluating Choices in a Virtual Community of Use,” Proc. Conf. Human Factors in Computing Systems, 1995.
[8] P. Resnick, N. Iakovou, M. Sushak, P. Bergstrom, and J. Riedl, “GroupLens: An Open Architecture for Collaborative Filtering of Netnews,” Proc. 1994 Computer Supported Cooperative Work Conf., 1994.
[9] U. Shardanand and P. Maes, “Social Information Filtering: Algorithms for Automating ‘Word of Mouth’,” Proc. Conf. Human Factors in Computing Systems, 1995. .
[10] M. Balabanovic and Y. Shoham, “Fab: Content-Based, Collabora- tive Recommendation,” Comm. ACM, vol. 40, no. 3, pp. 66-72, 1997.
[11] W.W. Cohen, R.E. Schapire, and Y. Singer, “Learning to Order Things,” J. Artificial Intelligence Research, vol. 10, pp. 243-270, 1999.
[12] Y. Freund, R. Iyer, R.E. Schapire, and Y. Singer, “An Efficient Boosting Algorithm for Combining Preferences,” Proc. 15th Int’l Conf. Machine Learning, 1998.
[13] R. Jin, L. Si, and C. Zhai, “Preference-Based Graphic Models for Collaborative Filtering,” Proc. 19th Conf. Uncertainty in Artificial Intelligence (UAI 2003), Aug. 2003a.
[14] R. Jin, L. Si, C. Zhai, and J. Callan, “Collaborative Filtering with Decoupled Models for Preferences and Ratings,” Proc. 12th Int’l Conf. Information and Knowledge Management (CIKM 2003), Nov. 2003b.
[15] Adomavicius, Gediminas, and Alexander Tuzhilin. "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions." Knowledge and Data Engineering, IEEE Transactions on 17.6 (2005): 734-749.