Applying Matrix Factorization to Recommendation System

Anonymous Author(s)
Affiliation
Address
email

Abstract

In recenet years, specially after the success of Netflix's grand prize challenge, Matrix Factorization has become the dominant tachnique for recommendation system for it's ability to handle latent factors and also to accomodate additional information like biases, temporal dynamics and confidence level. In this project, matrix factorization model is used for recommending music to users based on their listening history and habit. And also an implementation of Stochastic Gradient Descent algorithm was compared against two state-of-the-art matrix factorization algorithm namely Alternating Least Squares (ALS) and Multiplicative Updates Algorithm.

1 Introduction

The purpose of a recommendation system is to generate meaningful recommendations to a collection of users for items or products that might interest them. As modern consumers are inudated with choices, providing personalized recommendations that suits a user's taste has become significantly important. Suggestions for books on Amazon, or movies on Netflix are real-world examples of industry-strength recommender systems which have proven tremedeously influencing for the success of the respective companies. The domain and particluar characteristics of the data available guide the design of such recommendation systems. As an example, movie watchers on Netflix frequently provide ratings on a scale of 1 (disliked) to 5 (liked) which records the quality of interactions between users and items.Moreover, the system may have access to user-specific and item-specific profile attributes such as demographics and product descriptions, respectively. The way recommender systems analyze these data sources to develop notions of affinity between users and items differ from one another in different systems. **Collaborative Filtering** systems analyze historical interactions whereas **Content based Filtering** systems are based on profile attributes. Although two primary areas of collaborative filtering are **neighborhood methods** and **latent factor models**— the Netflix Prize competition has demonstrated , the matrix factorization models are superior to classic nearest-neighbor techniques for producing product recommedations, allowing the incorporation of additional information such as implicit feedback, temporal effects and confidence levels [1].

2 Matrix Factorization Methods

Matrix factorization is basically a latent factor model. In its basic form, matrix factorization characterizes both items and users by vectors of factors inferred from item rating patterns. High correspondance between item and user factors leads to a recommendation. These methods have become popular in recent years by combining good scalabilty with predictive accuracy [1]. In addition, they offer much flexibilty for modeling real life situations.

Recommender systems rely on different types of input data, which are often placed in a matrix with one dimension representing users and the other dimension representing items of interest. The most convenient data is high quality **explicit feedback**, which includes explicit input by users regarding their interest in products. As an example, Netflix collects star ratings for movies. These ratings can be thought of explicit feedback from the user. Ususally, explicit feedback comprises a sparse matrix, since single user is likely to have rated a small percentage of possible items.

One advantage of matrix factorization is that it allows incorporation of additional information. When explicit feedback is not available, recommender systems can infer user preference using implicit feedback, which indirectly reflects opinion by observing user behavior including purchase, search patterns or even mouse movements. Implicit feedback usually denotes the presence or absence of an event , so it is typically represented by a densely filled matrix.

3 Mathematics Behind Matrix Factorization

It can be said that 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 product in that space. Accordingly, each item *i* is associated with a vector $q_i \in R^f$ and each user u is associated with a vector $p_u \in R^f$. For a given item *i*, the element of q_i measure the extent to which the item possesses those factors whereas 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. The resulting dot product, ${q_i}^T p_u$, captures the interaction between the 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

$$\hat{r}_{ui} = {q_i}^T p_u \tag{1}$$

The major challenge here is computing the mapping of each item and user to factor vectors. After the recommender system completes the mapping, it can easily estimate the rating a user will give to any system by using Equation 1.

Such a model is closely related to singular value decomposition (SVD), a well-established technique for identifying latent semantic factors in information retrieval. Applying SVD in the collaborative filtering domain requires factoring the user-item rating matrix. This often raises difficulties due to the high portion of missing values caused by sparseness in the user-item ratings matrix. Conventional SVD is undefined when knowledge about the matrix is incomplete. Moreover, carelessly addressing only the relatively few known entries is highly prone to overfitting.[1]

Previous systems relied on imputation to fill in the missing ratings and make the rating matrix dense[2] which can be very expensive as it significantly increases the amount of data. Moreover, inaccurate imputation might distort the data considerably. So, more recent works [3] suggested modeling directly the observed ratings only , while avoiding overfitting through a regularized model. To learn the factor vectors (p_u and q_i), the system minimizes regularized squared error on the set of known ratings:

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

Here, K is the set of the user- item (u , i) pairs for which r_{ui} is known (i.e. the training set).

The system learns the model by fitting the previously observed ratings to generalize those previous ratings in a way that predicts future, unknown ratings. Thus the system should avoid the overfitting the observed data by regularizing the learned parameters, whose magnitudes are penalized. The constant λ controls the extent of regularization and is usually determined by cross-validation.

4 Methodology

In this scope, I've experimented with the three approaches to minimize Equation (2) . They are Stochastic Gradient Descent, Alternating Least Squares (ALS) and Multiplicative Update. Mat-

lab code was written for Stochastic Gradient Descent to compare with state-of-the art algorithm ALS and Multiplicative Update provided with the Matlab library function **nnmf** available for *Non-Negative Matrix Factorization*. The following section explains in brief how these three algorithm works:

4.1 Stochastic Gradient Descent

Simon Funk popularized a stochastic gradient descent optimization of Equation 2 [1] wherein the algorithm loops through all the ratings in the training set . For each given training case, the system predicts r_{ui} and computes the associated prediction error

$$e_{ui} = r_{ui} - {q_i}^T p_u$$

Then it modifies the parameters by a magnitude proportional to γ in the opposite direction of gradient , yielding :

$$q_i = q_i + \gamma(e_{ui} * p_u - \lambda * q_i)$$
$$p_u = p_u + \gamma(e_{ui} * q_i - \lambda * p_u)$$

This is a very pouplar approach [1] which combines implementation ease with a relatively fast running time.

4.2 Alternating Least Squares

Because of the convexity of Equation 2 (as both q_i and p_u are unknowns) if we fix one of the unknowns, the optimiztion problem becomes quadratic and can be solved optimally. Thus ALS techniques rotate between fixing the q_i's and fixing the p_u's. When all p_u's are fixed, the system re-computes the q_i's by solving the least squares problem and vice versa. Hence it is called Alternating Least Squares. This alternating ensures that each step decreases Equation 2 until convergence. [1]

In general stochastic gradient descent is easier and faster than ALS, but ALS is favourable at least in 2 cases [1]. The first case is when the system can use paralleization. In ALS, the system computes each q_i independently of the other item factors and computes each p_u independently of the other user factors. This gives rise to potentially massive prallelization of the algorithm. The second case is for the system centered on the implicit data(like the current implementation). As the training set cannot be considered sparse, looping over every single training case– as gradient descent does – would not be that practical. ALS can handle these cases efficiently.

4.3 Multiplicative Updates

Given a non-negative matrix V, the problem of finding efficient non-negative matrix factors W and H can be described by the two alternative formulation of the optimization problems for NMF [4] :

Problem 1 Minimize the eucledian distance $\| V - WH \|^2$ with respect to W and H subject to the constraints $W, H \geq 0$

Problem 2 Minimize the divergence $D(V \parallel WH)$ with respect to W and H , subject to the constraints $W, H \geq 0$

`Multiplicative Update Rules` come with a good compromise between speed and ease of implementation for solving problems 1 and 2 [4] :

Theorem 1 The eucledian distance $\| V - WH \|$ is non-increasing under the update rules

$$H_{au} \leftarrow H_{au} \frac{(W^T V)_{au}}{(W^T WH)_{au}}$$

$$W_{ia} \leftarrow W_{ia} \frac{(VH^T)_{ia}}{(WHH^T)_{ia}}$$

The Eucledian distance is invariant under these updates if and only if W and H are at a stationary point of the distance.

Theorem 2 The divergence $D(V \parallel WH)$ is non-increasing under the update rules

$$H_{au} \leftarrow H_{au} \frac{\sum_i W_{ia} V_{iu} / (WH)_{iu}}{\sum_k W_{ka}}$$

$$W_{ia} \leftarrow W_{ia} \frac{\sum_u H_{au} V_{iu} / (WH)_{iu}}{\sum_v H_{av}}$$

The divergence is invariant under these updates if and only if W and H are at a stationary point of the divergence.

5 Dataset

The dataset I used famously called **last-fm** dataset is from the Last.FM music website (http://www.last.fm/) that allows users to tag bands, albums and songs. This dataset contains user-artist pairs and thus represent an artist recommendation task (= item recommendation): suggest new, interesting bands for the user to listen to, based on his or her past listening habits. Each band has been listened to at least 20 times. The users does not give any explicit rating – so the recommendation task is performed by implicit feedback based on the listening history and habit of the users.

6 Evaluation

The typical measure of the performance of the recommendation systems are `predictive accuracy metrics` [5] where the predicted ratings are directly compared to actual user ratings. The most commonly used metric in the literature is **Mean Absolute Error (MAE)** – defined as the average absolute difference between the predicted ratings and actual ratings, given by :

$$MAE = \frac{\sum_{u,i} |p_{u,i} - r_{u,i}|}{N}$$

where $p_{u,i}$ is the predicted rating for user u on item i , $r_{u,i}$ is the actual rating and N is the total number of ratings in the test set.

Another commonly used metric is **Root Mean Squared Error (RMSE)** which puts emphasis on larger absolute error and is given by:

$$RMSE = \sqrt{\frac{\sum_{u,i} (p_{u,i} - r_{u,i})^2}{N}}$$

Both of these measures were used to measure the performance of the algorithms applied.

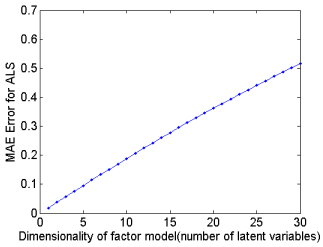


Fig 1: MAE for Alternating LS

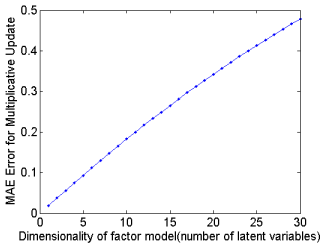


Fig 2 : MAE for Multiplicative Up.

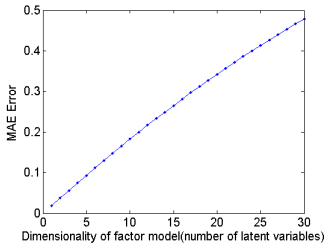
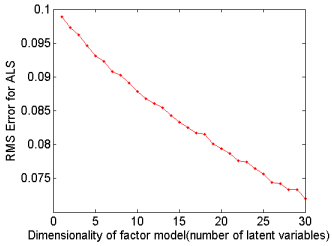
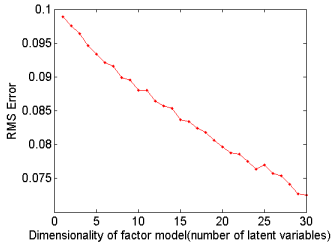


Fig 3: MAE for Gradient Dsnt.





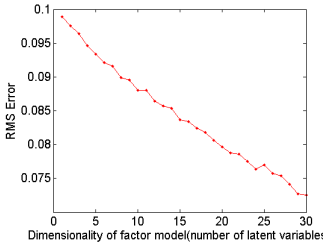


Fig 1:RMSE for Alternating LS Fig 2 :RMSE for Multiplicative Up. Fig 3:RMSE for Gradient Dt.

From the above figures we can see that all three algorithms show matching performance in Mean Absolute Error(MAE) and Root Mean Squared Error (RMSE). But stochastic gradient descent takes a long time to converge as it particularly performs well for sparse matrix –but for dense matrix (as in our case) it has to loop over every single case. Also the stopping condition consiedered (0.001) was very small so that it does not miss any minima and does not oscillate thereby. But this also contributed the runnung time.

The problem of implementing matrix factorization algorithm in Matlab is that the software cannot deal with sparsity of the matrix which imputes data with default values and also makes the matrix dense – so actually can't take the advantage of the sparsity of the matrix.

7 Conclusion

The adaptive nature of matrix factorization along with its ability to manipulate latent factors has given it an edge over all other models for recommender systems. It can incorporate user and item biases successfully to improve its predictive performance. Like some items are generally popular than others, some users are more generous or critical than others– these information can be incorporated successully in the matrix factorization framework. To handle the famous cold start or new user problem –this model can efficiently use implicit feedback using the user's behavioral pattern. Different items popularity might change over time – so considering temporal dynamics can improve the recommender's perofrmance significantly and matrix factorization can be extended to handle these temporal dynamics too. Moreover, the matrix factorization model can readily accept varying confidence levels, which let it give less weight to less meaningful observations and thus improving the systems overall performance. With all these defining characteristics matrix factorization has truly become de-facto standard for recommendation system now-a-days.

References

[1] Koren, Y.,Bell,R. & Volinsky, C. "Matrix Factorization Techniques for Recommender Systems", *In Proceedings of IEEE Computer. 2009*, pp.30-37.

[2] B.M. Sarwar et al. "Application of Dimensionality Reduction in Recommender System– A Case Study"*Proc. KDD Workshop on Webmining for e-Commerce: Challenges and opportunities(WebKDD)*, ACM Press,2000.

[3] Koren,Y."Factorization Meets the Neighborhood: A Multifaceted Collaborative filtering model" *Proc. 14th ACM SIGKDD International Conference of Knowledge Discovery and Data Mining* , ACM Press , 2008, pp. 426-434.

[4] Lee,D. & Seung, S. "Algorithms for Non-negative Matrix Factorization", *In NIPS*,MIT Press, 2001, pp.556-562.

[5] Herlocker, J.L. et al. "Evaluating collaborative filtering recommender systems ", *ACM Transactions on Information Systems*, 22(1),5-53.