Project Report of Movie Recommendation

林泽冰,5120309085 杨光,5120309016 林洋,5120309005

May 29, 2015

1 Project Summary

1.1 Project Requirement

In this project, we need to implement a movie recommendation system and give predictions.

1.2 Dataset Description

The dataset contains a training set (training.txt) and a test set (test.txt).
Every line of training.txt corresponds to a rating on a movie by a user on a specific day.
UserId,MovieId,DateId,Rating
- UserId ranges from 0 to 99999
- MovieId ranges from 0 to 17769
- DateId ranges from 0 to 5114 (x + 1 is the day after x)
- Rating ranges from 1 to 5

test.txt is similar to training.txt. But the all the ratings are removed and you need to give predictions for them.
UserId,MovieId,DateId

1.3 Preliminaries

We are given ratings about m users and n items(henthforth, interchangeable with "movies"). We reserve special indexing letters for distinguishing users from items: for user u, v and for items i,j. We use t for time(or date). A rating $r_{ui}(t)$ indicates the preference by user u of item i at day t, where high values mean stronger preferences. When the day of rating is irrelevant, we use the short notation r_{ui}. We distinguish predicted rating from known ones, by using the notation of $\widehat{r}_{ui}(t)$ for the predicted value of $r_{ui}(t)$.

1.4 Overview

We first adopted biased item-item collaborative filtering to predict, using pearson correlation to measure similarities between items. However, we found that it was too time-consuming since the spare user-item matrix cannot fit into memory. What's worse, after running for 16 hours, the predict result is discouraging: the overall RMSE is approximately 1.01. Thus, we turned to latent factor model. We mainly referred to [1] and [2], and implemented SVD, SVD++ and timeSVD++ in C++ under windows.

Note that we use some of the metrics of C++11, so if you are running under windows, make sure that you enable -std=c++11(or -std=c++0x); if you are running under ubuntu, make sure that the version of g++ is 4.6 or above.

2 SVD

2.1 Theory

Matrix factorization models map both users and items to a joint latent factor space of dimensionality f. The latent space tries to explain ratings from user feedback. To be specific, each item is associated with a vector $q_i \in \mathbb{R}^f$, and each user is associated with a vector $p_u \in \mathbb{R}^f$. 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 which user has in items that are high on the corresponding factors. The resulting dot product, $q_i^T p_u$, captures the interaction between user u and item i, i.e., the overall interest of the user in characteristics of the item. The final rating is created by adding the baseline predictor, which takes the biases of users and items into account:

$\widehat{r}_{ui}(t) = \mu + b_i + b_u + q_i^T p_u$

(μ: Overall mean rating, b_i: bias for item i, b_u: bias for user u)

In order to learn the model parameters(b_u,b_i,p_u and q_i),, we have to minimize the regularized squared error:

$\min \sum_{(u,i)\in K} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda_4 (b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2)$

The constant λ_4, which controls the extent of regularization, is usually determined by cross validation.

2.2 Implementation

We use an easy stochastic gradient descent optimization method, which loops through all the ratings in the training data. For each given rating r_{ui}, a prediction $\widehat{r}_{ui}(t)$ is made, and the associated prediction error $e_{ui} = r_{ui} - \widehat{r}_{ui}(t)$ is computed. For a given training case r_{ui}, we modify the parameters by moving in the opposite of the gradient, yielding:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_i)$
- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda_4 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda_4 \cdot p_u)$

According to [2], we set $\gamma = 0.005, \lambda_4 = 0.02$. And we set the factors to be 200.

main.cpp first generates the cross validation file named "cross.txt", and "train.txt" used for training. After intensive tests, we randomly choose one percent of "training.txt" as "cross.txt", and the rest as "train.txt". We run the algorithm on "train.txt" and use "cross.txt" to calculate the RMSE. When the RMSE stops decreasing, we stop training. **The cross validation part is quite essential, otherwise you will end up in overfitting.**

The implementation is just intuitive and more information is provided in the codes. We implement a class named "SVD", and the parameters are set during its initialization. **SVD::predictScore()** and **SVD::Train()** are implemented according to the rules specified above. Note that the data can fit into memory to reduce time cost.

2.3 Result

The result is pretty good. It takes approximately 20-30 iterations to converge. On the test site, we achieved RMSE 0.91564.

3 SVD++

3.1 Theory

Prediction accuracy is improved by considering also implicit feedback, which provides an additional indication of user preferences. This is especially helpful for those users that provided much more implicit feedback than the explicit one. To be specific, we can capture a significant signal by accounting for which items users rate, regardless of their rating value.

To this end, a second set of item factors is added, relating each item i to a factor vector $y_i \in \mathbb{R}^f$. Those new item factors are used to characterize users based on the set of items that they rated. The exact model is as follows:

$\widehat{r}_{ui}(t) = \mu + b_i + b_u + q_i^T(p_u + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j)$

The set $R(u)$ contains the items rated by user u.

A user is modeled as $(p_u + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j)$, and $|R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j$ indicates the perspective of implicit feedback, and the sum of y_j is normalized by $|R(u)|^{-\frac{1}{2}}$.

3.2 Implementation

Still, we use stochastic gradient decent to minimize the associated regularized squared error function. We loop through all the ratings grouped by user computing:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_i)$
- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot (p_u + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j) - \lambda_6 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda_6 \cdot p_u)$
- $\forall j \in R(u)$:
  $y_j \leftarrow y_j + \gamma \cdot (e_{ui} \cdot |R(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_6 \cdot y_j)$

We set $\gamma = 0.007, \lambda_5 = 0.005, \lambda_6 = 0.015$ according to [2]. Note that it's beneficial to decrease the step sizes (the γ's) by a factor of 0.9 after each iteration.

The framework is basically the same as the SVD model. However, there are some tricks to reduce time cost. For example, when you are updating q_i, it's beneficial to calculate $\sum_{j \in R(i)} y_j$ in advance, otherwise you have to recalculate it for every q_i, and that would be a catastrophe. Similarly, when you are updating y_j, though we are supposed to update at every rating, we choose to update at every user, otherwise the time cost would be terrible. It's a kind of approximation inspired by [3],but it greatly improves the training time.

Another things worth mentioning is that it's possible that $\|R(u)\| = 0$. When this happens, we ignore the implicit feedback.

3.3 Result

The result is encouraging. The iterative process runs for around 30 iterations until convergence. On the test site, we achieved RMSE 0.91133 with 50 factors.

It's worth mentioning that we also tried to take items' implicit feedback into consideration:

$\widehat{r}_{ui}(t) = \mu + b_i + b_u + p_u(q_i^T + |R(i)|^{-\frac{1}{2}} \sum_{j \in R(i)} y_j)$

We tried it and also tried to train in turn by the two svd++ models, and tried to blend the result of the two. However, it turned out that they didn't overperform 0.91133. It's quite natural, since items can't select uses, but users can select items.

4 timeSVD++

4.1 Theory

Much of the temporal variability is included within the baseline predictors, through two major temporal effects. The first addresses the fact that an item's popularity may change over time. The second addresses that temporal effect allows users to change their baseline ratings over time. See the figures below [1]:

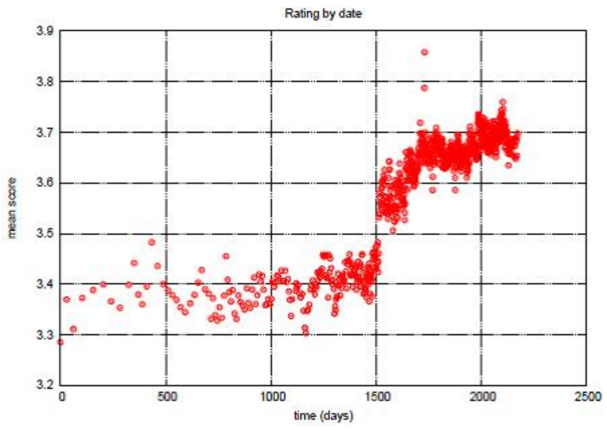

Figure 1: The average rating made a sudden jump in early 2004.

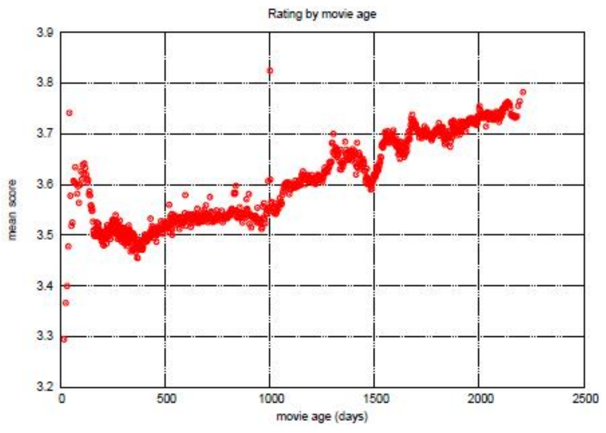

Figure 2: Rating tend to increase with the movie age at the time of rating.

To address these two, we treat the item bias b_i and user bias b_u as a function of time. A template for a time sensitive baseline predictor is:

$b_{ui} = \mu + b_u t_{ui} + b_i t_{ui}$

For time-changing item biases $b_i(t)$, we found it adequate to split the item biases into time-based bins, using a constant item bias for each time period. In our implementation, we choose 30 bins spanning all days in the dataset. A day t is asscociated with an integer $Bin(t)$, such that movie bias is split into stationary part and a time changing part:

$b_i(t) = b_i + b_{i,Bin(t)}$

Our simple modeling on the temporal effects of users is to capture a possible a gradual drift of user bias. For each user u, we denote the mean date of rating bu t_u, and the associated time deviation of this rating is defined as:

$dev_u(t) = sign(t - t_u) \cdot |t - t_u|^\beta$

$|t - t_u|$ measures the number of days between dates t and t_u. β is set to 0.4 according to [2]. We introduce a single new parameter for each user called α_u so that the time-dependent user-bias is:

$b_u(t) = b_u + \alpha_u \cdot dev_u(t)$

In the movie rating dataset we have found that multiple ratings a user gives in a single day. The effect may reflect the mood of the user that day, the impact of ratings given in a single day on each other, or changes in the actual rater in multi-person accounts. To address such short-lived effects, we assign a single parameter per user and day, denoted by $b_{u,t}$. It serves as an additive component within the previously described schemes.

In our implementation, we adopt the following rule of the linear+ model to model the drifting user and item bias:

$\widehat{r}_{ui}(t) = \mu + b_i + b_u + p_u(q_i^T + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j) + \alpha_u \cdot dev_u(t_{ui}) + b_{u,t_{ui}} + b_{i,Bin(t_{ui})}$

4.2 Implementation

We loop through all the ratings grouped by user computing:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_i)$
- $b_{u,t} \leftarrow b_{u,t} + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_{u,t})$
- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot (p_u + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j) - \lambda_6 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda_6 \cdot p_u)$
- $b_{i,Bin(t_{ui})} \leftarrow b_{i,Bin(t_{ui})} + \gamma \cdot (e_{ui} - \lambda_7 \cdot b_{i,Bin(t_{ui})})$

- $\alpha_u \leftarrow \alpha_u + \gamma_2 * (e_{ui} * dev_u(t_{ui}) - \lambda_8 * \alpha_u)$
- $\forall j \in R(u)$:
  $y_j \leftarrow y_j + \gamma \cdot (e_{ui} \cdot |R(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_6 \cdot y_j)$

After tuning the parameters for days, our parameters are set to be: $\lambda_5 = 0.005, \lambda_6 = 0.015, \lambda_7 = 0.08, \lambda_8 = 0.0004, \gamma = 0.007, \gamma_2 = 0.00001$ and the number of bins are 30.

4.3 Result

We found that the existence of $b_{u,t}$ is essential. Without it, the RMSE only dropped about 0.001 compared with SVD++. However, after including $b_{u,t}$, we achieved RMSE 0.904791. It takes about 30 iterations to converge.

5 Conclusion

Through this project, by implementing latent factor models ourselves, we get a deeper understanding of the gradient descent method, along with powerfulness of the latent factor model. By considering the temporal effects, we greatly improved our results, and realized that analysing the datasets is quite vital vital, through which we can dig more information.

Upon the deadline, we rank 1st and beat all other teams. Thanks to the help of our professor and the TA.

References

[1] Koren Y. Collaborative filtering with temporal dynamics[J]. Communications of the ACM, 2010, 53(4): 89-97.

[2] Y. Koren and R. Bell, “Advances in Collaborative Filtering,” in Recommender Systems Handbook, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Springer, 2011, pp. 145 – 186.

[3] Chen T, Zheng Z, Lu Q, et al. Feature-based matrix factorization[J]. arXiv preprint arXiv:1109.2271, 2011.