Implementation and Analysis of Collaborative Filtering Algorithm on MovieLens Latest Small Dataset

Xinqi Zhu 5150297, Ryan Mckay

[1]School of Computer Science and Engineering – University of New South Wales

Abstract. *We study the problem of implementing the recommender system on user-item rating system, i.e. user-movie rating system. We focus on the algorithm of collaborative filtering of latent factor model and KNN model. In this report, we show that how our system is implemented and how to get a reasonable prediction accuracy based on experiments. Different methods are used to evaluate the performance of our recommender system.*

1. Introduction

Recommender system has been widely used over the Internet and many Internet giants have their own version of recommender system to guide customers toward products they are more likely to be interested in. One of the most successful recommendation algorithms is collaborative filtering.

2. Data Preparation

3. Implementation

Our implementation mainly consists of five parts, including basic Collaborative Filtering latent factor models, k-nearest-neighbor model, k-fold cross validation, precision-recall curve and visualization.

3.1. Collaborative Filtering Algorithm

There are two primary areas of collaborative filtering: neighborhood methods and latent factor models[1]. Our realization of the latent factor model called matrix factorization(MF) and k-nearest-neighbor for neighborhood method.

3.1.1. Latent Factor Model

This algorithm will model the user-movie rating matrix as inner product of a user matrix and a movie matrix. Each movie is represented as a vector $X_i \in \mathbb{R}^f$ and each user is associated with $\theta_j \in \mathbb{R}^f$. The approximate rating by user j on movie i is:

$$\hat{r}_{ij} = X_i^T \cdot \theta_j$$

This implementation require the determination of the number of features f, which is used to represent the features of a movie and the preference of a user on these features. In Experimental Evaluation section we discuss the number of feature selection.

In order to get the proper X and θ matrix, we use the definition of cost function from [1]:

$$J = \frac{1}{2} \sum_{(i,j):t(i,j)=1} (X_i^T \cdot \theta_j - r_{ij})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{f} X_{ik}^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{f} \theta_{jk}^2$$

where $t(i, j)$ means movie i has been rated by user j. λ is the regularization coefficient and should be determined by experiment.

The gradient of X and θ:

$$\frac{\partial J}{\partial X_{ik}} = \sum_{j:t(i,j)=1} (X_i^T \cdot \theta_j - r_{ij})\theta_{jk} + \lambda X_{ik}$$

$$\frac{\partial J}{\partial \theta_{jk}} = \sum_{i:t(i,j)=1} (X_i^T \cdot \theta_j - r_{ij})X_{ik} + \lambda \theta_{jk}$$

Then we use the predefined optimization method of Conjugate Gradient from scipy library to train our model. In order to do so, we compress the X and θ matrix into a vector to fit the minimization function.

When doing prediction, we get the inner product of X and θ matrix to get $\hat{r}$ matrix. Then we replace each value in $\hat{r}$ by the nearest rating level (0.5, 1.0, 1.5 ...) and compare it with the true rating matrix r (details in section 5).

3.1.2. K-Nearest-Neighbor Model

3.2. K-fold Cross Validation

We use k-fold cross validation to determine the proper number of features f, the regularization coefficient λ and do any other debugging.

By using the KFold function of sklearn's model_selection library, we can easily split the training set into k folds and use any of them as validation set. In our implementation, we take $k = 5$ and use the mean of each fold's MSE(mean square error) and MAE(mean absolute error) as final training set's and CV's MSE and MAE.

4. Experimental Evaluation

We randomly split the raw data of user-movie rating into training set(80%) and test set(20%), then do experiments with k-fold cross validation with $k = 5$. The experiments include determination of the number of features f, regularization coefficient λ, getting learning curve and ROC(receiver operating characteristic) curve.

Figure 1 and Figure 1 show the experiments to get the proper f and λ and ends up with $f = 105$ and $\lambda = 1.3$. We can see when λ is larger than 1.3 the MSE on CV set is increasing and likely to be underfitting(high bias). On the contrary, low λ makes the system overfitting(high variance). We set $f = 105$ at the elbow point because larger f takes more time to train but no obvious improvement on result, and lower f is not mature enough.

Figure 1 shows the learning curve under three conditions: underfitting($\lambda = 5$), normal($\lambda = 1.3$), and overfitting($\lambda = 0.2$).

Figure 1 shows the ROC curve of our recommender system. The confusion matrix is defined based on the assumption that the truth rating beyond a threshold should be regarded as true positive and below as true negative[2]. For prediction, we maintain a top rating list, i.e. top 100 rating movies to be predicted positive and rest to be predicted negative. We test three thresholds, 3, 4 and 4.5 stars. We draw the curve by changing the size of the top N list.

Figure 1. A typical figure

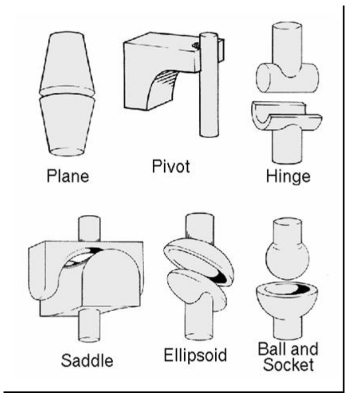

Figure 2. This figure is an example of a figure caption taking more than one line and justified considering margins mentioned in Section ??.

4.1. Results

For latent factor model, our system reaches an mean square error of 0.9 and mean absolute error of 0.5. The average accuracy of correct prediction on rated movies is 70%. This is an amazing ratio because most of the user's interaction with movies has been correctly predicted. The ROC curve shows that the prediction is much better than randomly guess.

5. Future Work

6. Conclusion

References

[1] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems, 2009.

[2] Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2):81–173, 2011.