Homework 9: Recommender Systems

Vidhan Agarwal(vidhana), Xin Wang(xinw3)

1 Problem Statement

Given a dataset containing users and their ratings of some movies, we are supposed to predict the rating if provided a pair of userid and movieid. In this way, we can make recommendations to users. We only rely on the previous behaviour of users, so we choose to use *Collaborative Filtering* algorithm.

The data we are provided is *explicit feedback* in recommender systems, causing a sparse matrix. In *Collaborative Filtering*, *Latent Factor Models* is a good way to deal with this situation. Assuming we have N users and M movies, D is the number of the latent factors. Let U, a $D \times N$ matrix, denotes the user coefficient matrix, and V, a $D \times M$ matrix, denotes the movie coefficient matrix. *Latent Factor Models* can be represented by $\hat{R} = U^T V$[2].

We choose to use *Matrix Factorization* to implement *Latent Factor Models*, which is suited for sparse matrix because it's domain free and allows incorporation of additional information[1].

2 Data Pre-processing and Feature Engineering

First we need to clean our data because there are some missing fields in the training dataset. Then we split the data into two parts: training set (95%)and validation set(5%). The reason why we use a validation set is to avoid overfitting. When we increase the number of latent factors D, the training RMSE is always decreasing. After adding a validation set, when D is reached a certain number, the validation RMSE decreases first and then increases, which means it is overfitting and we need to tuning down D. Our D is chosen between 1 to 20. In our model, when D is in the range of [5, 8], the best results can be acquired.

In *Matrix Factorization*, We are not supposed to use missing ratings, so we construct two dictionaries(Python). One contains movieids that each user has rated. The other contains userids that one movie has been rated by. We use those two dictionaries to update U and V in learning algorithm.

3 Model Implementation

Our objective function is *Squared Error*, the goal of our algorithm is to find U and V that give the minimum Squared Error:

$$U, V = \underset{U,V}{argmin} \sum_{(i,j)} w_{ij}(\hat{R}_{ij} - R_{ij})^2$$

where

$$w_{ij} = \begin{cases} 1 & R_{ij} \text{ is known} \\ 0 & R_{ij} \text{ is unknown} \end{cases}$$

Our learning algorithm is *Alternating Least Squares*(ALS). In every iteration, we first fix V to update U, then we use the newly-updated, fixed U to update V. We've found that it is more intended to overfit the training set. So we add a regularized term to the objective function to penalize the complexity. The regularized objective is as the following:

$$U, V = \underset{U,V}{argmin} \sum_{(i,j)} w_{ij}(\hat{R}_{ij} - R_{ij})^2 + \lambda(||U_i||^2 + ||V_j||^2)$$

We solve the equation by setting its derivative to 0 [4], then get:

$$U_i = (V \times w_i \times V^T + \lambda I)^{-1} \times V \times w_i \times r_i$$
$$V_j = (U \times w_j \times U^T + \lambda I)^{-1} \times U \times w_j \times r_j$$

The initial values for U and V are average ratings. We keep updating the parameters until the U and V are converging.

4 Results Analysis

In the very first version, our RMSE was 1.7 by using SGD. Then we changed to ALS, the RMSE dropped down to 1.07. We analyzed our code and figured out that we actually implemented SVD rather than Matrix Factorization because we didn't ignore missing ratings. Instead, we used random numbers between 1 and 5 to replace the empty cells in the ratings matrix. Then we tried imputation with the mean of the ratings. The performance did get improved, from 1.07 to 0.99, but still not satisfying. Later on, we figured out that we are not supposed to use the missing cells. After changing the implementation to only use the rated movies, we got 0.86 RMSE.

5 Thinking Process

We discussed the paper Probabilistic Matrix Factorization [2] to see if we can use this model. It's fit for the datasets that are sparse and imbalanced. At first we implemented the unconstrained PMF. We also compared different learning algorithm such as SGD, ALS and even DSGD. As Aberger stated in [3], most of the cases, SGD is faster, but ALS performs better in MovieLens dataset and often scales better. For DSGD, it's suited for very large datasets like several hundreds Gigabytes.

To avoid making predictions outside of the range of valid ratings values, we tried to pass the dot product through the logistic function $\sigma(x) = \frac{1}{1+exp(-x)}$, and also normalized the ratings using $t(x) = \frac{x-1}{K-1}$[2], where K is the maximum ratings(5 in our case).

Later on, we figured out that we are under the wrong track, so we changed back to classic ALS and got satisfied performance.(RMSE 0.86)

6 Collaboration

6.1 Workload Sharing

Vidhan Agarwal: Code optimization and implementation, report.

Xin Wang: Code implementation, report.

6.2 Did you give any help to other students?

No

6.3 Did you receive any help from other students?

No

7 Time Spent

60 hours.

References

[1] Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 42.8 (2009).

[2] Salakhutdinov, Ruslan, and Andriy Mnih. "Probabilistic Matrix Factorization." *Nips*. Vol. 1. No. 1. 2007.

[3] Aberger, Christopher R. "Recommender: An Analysis of Collaborative Filtering Techniques."

[4] https://datasciencemadesimpler.wordpress.com/tag/alternating-least-squares/