CMPT-741 Course Project : Recommender System

Amandeep Singh Kapoor askapoor@sfu.ca
Lakshayy Dua ldua@sfu.ca
Shariful Islam msislam@sfu.ca

1 Introduction

With huge amount of available products and consumers with different types of needs, any business, specially any e-commerce business would like to match consumers with the most appropriate product to enhance user satisfaction. Most of the e-commerce businesses have become interested in recommender systems to provide personalized recommendations according to users need.

Problem definition

In this project we build a Recommender System (RS) for Yelp. The problem is to predict user rating (in the range of 1-5) for a given user-item pair based on historical ratings provided by many users. The performance of the recommender system is evaluated using Root Mean Square Error (RMSE) of the predicted rating of the test set user-item pair with respect to known ratings. The goal of this project is to minimize the RMSE on the test set.

Dataset

For this problem we use a Yelp dataset, which is a collection of 2038130 user-item pair and the corresponding rating. The dataset contains five columns `train_id`, `user_id`, `business_id`, `rating` and `date`. The rating is between 1 and 5. The date column represents the date on which the review was given.

2 Methodology

Recommender systems are broadly based on one of two basic strategies [1], such as (a) content filtering approach, and (b) collaborative filtering approach. In this project we focus on the collaborative filtering approach, which depends only on the past behaviour and does not require any domain specific knowledge. There are two popular ways to implement collaborative filtering, namely, (a) neighbourhood method and (b) latent factor method. Latent factor methods explains the ratings by finding latent factors from both the items and the users, which is derived from the original dataset [1]. In this project we do not consider implementing neighbourhood method, and focus on the variations of latent factor method.

One of the most successful implementation of latent factor method can be done using matrix factorization [1]. We focus on a few different versions of matrix factorization and observe their performance on the test set. At the end, we also use a library named *surprise* [2] which implements Singular Value Decomposition algorithm. As the test set rating is not made open to us, for testing performance and parameter tuning we use k-fold cross validation. The methods we have applied are discussed in the following section.

2.1 Matrix Factorization Method

Matrix factorization method maps user and item to a factor space of N dimensions. Each item i can be associated with a vector q_i, and each user u can be associated with a vector p_u. Both q_i and p_u are of dimension N, optimal value of which can be determined by parameter tuning. The goal of

the matrix factorization technique is to learn the value of q_i and p_u using some machine learning technique. By knowing these two vectors we can determine estimated rating of item i given by user u according to Equation 1.

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

We want to learn the values of q_i and p_u using the known ratings. As we are trying to minimize the RMSE on the test set, we need to minimize the sum of the squared error over the training data. Considering the regularization term to avoid overfitting for the train data the final equation we get is represented by Equation 2.

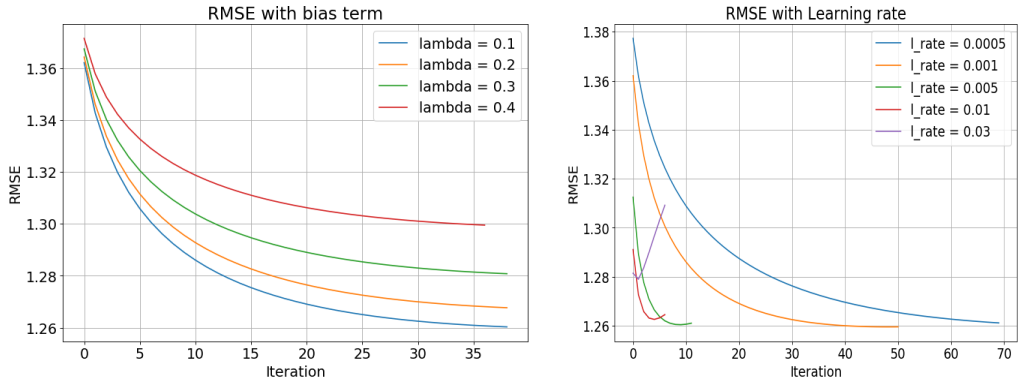
$$\min_{q^* p^*} \sum_{(u,i)\in\kappa} (r_{ui} - {q_i}^T p_u)^2 + \lambda(|q_i|^2 + |p_u|^2) \tag{2}$$

where, κ is the set of (u, i) pairs for which the rating is already known. Now we can learn the value of p^* and q^* by solving the previous equation. If we solve for p^* and q^* and apply gradient descent for updating them we can express the update equation by Equation 3 and 4

$$q_i \leftarrow q_i + \gamma.(e_{ui}.p_u - \lambda.q_i) \tag{3}$$
$$p_u \leftarrow p_u + \gamma.(e_{ui}.q_i - \lambda.p_u) \tag{4}$$

where, $e_{ui} = r_{ui} - {q_i}^T p_u$, is the error in the estimated rating and γ is the learning rate.

There are two popular approaches to implement this learning algorithm, namely, *stochastic gradient descent* and *alternating least squares* (ALS). We experiment with both of the algorithms to observer their performance.



(a) (b)

Figure 1: (a) RMSE on test set (made from full dataset) with different λ (b) RMSE on test set (made from full dataset) with different γ (learning rate)

Implementation

The key approaches to implement the learning algorithm is described as follows:

1. As the dataset is huge, and if we try to make a conventional user-item matrix, it does not fit into the memory. We exploit the fact that most of the user-item pair does not have a rating value. Thus, we use a sparse matrix to represent the user-item matrix. Which solves the memory problem. We use `sparse` from `scipy` library to implement this.
2. The optimal value of parameters λ and γ needs to be determined. As the ratings of the original test data is not made open, we split the full data into test set and training set to test our algorithm and use different values of the parameters to find the best set of values.

   Figure 1(a) shows RMSE on the synthetic test set with different values of λ. We see that as we decrease the value of λ, the value of RMSE on the test set decreases with number of iteration and reaches to saturation after around 35 iterations. Decreasing the value of λ

after .1 does not improve the RMSE significantly. Decreasing the value of λ too much on the other hand will nullify the effect of regularization term.

Figure 1(b) shows RMSE with different learning rate for $\lambda = .1$. We observe that if learning rate is too small it takes too many iteration to reach to saturation. On the other hand increasing the learning rate makes the learning very fast but the learning algorithm does not reach to the possible minimum value of the test set RMSE. $\gamma = .001$ seems like a optimum value for learning rate. However, for our best test result, we set our learning rate to be $.0035$.

3. Some of the ratings estimated by the algorithm goes over 5 and below 1, which we definitely know is wrong. To enhance the performance in terms of RMSE we convert all the ratings over 5 as 5 and all the ratings below 1 as 1. This pruning method improves the performance as reported in Section 3.
4. We also implemented alternating least square algorithm for the same sets of parameters. For this particular dataset, this method did not provide any improvement in terms of RMSE. However, that makes sense, because, according to [1], ALS works better when we have parallelization of the algorithm and in the case of implicit data.

2.2 Matrix Factorization with Bias Term

Using bias terms associated with different users and items can be incorporated into matrix factorization technique [1]. If a user u has bias b_u and an item has bias b_i we can think of a bias term for each user-item pair expressed by Equation 5

$$b_{ui} = \mu + b_i + b_u \tag{5}$$

where μ is the average rating over the whole dataset. Using similar argument, the estimated rating can be given by the following equation.

$$\hat{r}_{ui} = \mu + b_i + b_u + {q_i}^T p_u \tag{6}$$

Thus we need to estimate q_i, p_u, b_i and b_u using some machine learning technique. According to [1] to estimate the optimum value of all these parameters, we need to solve the following minimization problem.

$$\min_{q^* p^*} \sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_i - b_u - {q_i}^T p_u)^2 + \lambda(|q_i|^2 + |p_u|^2 + {b_u}^2 + {b_i}^2) \tag{7}$$

Solving Equation 7 for q_i, p_u, b_i, b_u provides us with the following update equations.

$$b_i \leftarrow b_i + \gamma.(e_{ui} - \lambda.b_u) \tag{8}$$

$$b_u \leftarrow b_u + \gamma.(e_{ui} - \lambda.b_i) \tag{9}$$

$$q_i \leftarrow q_i + \gamma.(e_{ui}.p_u - \lambda.q_i) \tag{10}$$

$$p_u \leftarrow p_u + \gamma.(e_{ui}.q_i - \lambda.p_u) \tag{11}$$

where $e_{ui} = r_{ui} - \mu - b_i - b_u - {q_i}^T p_u$. We can use Equation 8 - 11 to learn all the parameters using stochastic gradient descent method. Matrix factorization method with bias term improved the performance in terms of RMSE, which is reported in Section 3.

2.3 Matrix Factorization with Temporal Bias

We can use temporal bias for user and item to improve the performance of the learning algorithm [3]. In our experiment we tried to incorporate a few temporal parameters. Including a temporal parameter to the item bias term can be represented using the following equation.

$$b_i(t) = b_i + b_{i,Bin(t)} \tag{12}$$

Here, Bin represents the number of sets we have divided the time in our training dataset. We have included the time specific Item bias on an yearly basis and added it to the total item bias so that the Item bias will not be a constant function but a function that changes over time. Using Equation 12 we can model user-item pair bias with item temporal term as,

$$b_{ui}(t) = \mu + b_u + b_i + b_{i,Bin(t)}. \tag{13}$$

Equation 13 include the extended temporal effects on about Items popularity while predicting the overall training. This $b_{i,Bin(t)}$ is also parameter which would be learned through stochastic gradient descent technique.

Table 1: Performance on test set (Kaggle) using different algorithms and parameters

Algorithm	N	iteration	λ	γ	RMSE
Stochastic Gradient Descent	200	35	.3	.001	1.31693
Alternating Least Square	200	35	.3	.001	1.31693
Stochastic Gradient Descent	200	50	.3	.001	1.31476
Alternating Least Square	200	50	.3	.001	1.31476
Stochastic Gradient Descent	200	42	.2	.001	1.30441
Stochastic Gradient Descent (prune)	200	42	.2	.001	1.30435
Stochastic Gradient Descent (prune)	200	42	.1	.001	1.29958
Stochastic Gradient Descent (prune)	200	42	.07	.001	1.29912
Surprise recommender library	-	-	-	-	**1.29566**
Stochastic Gradient descent (with bias term)	200	60	.15	.0035	**1.29482**

2.4 Surprise Package

We also used a library named Surprise [2] to predict users rating. We achieved a RMSE score of less than 1.3.

Surprise makes use of `evaluate()` function which gives the results on one set of parameters given to the algorithm. To try the algorithm on different set of parameters, we used GridSearch to exhaustively try all the combination of most likely parameters. In the end it gave us the best combination for lowest RMSE.

We loaded our train and test set using `load_from_df()` method. We used a Reader object, and specified `rating_scale` parameter on scale of 1-5. The dataframe must have three columns, corresponding to the user (raw) ids, the item (raw) ids, and the ratings in respective order. Each row thus corresponds to a given rating.

SVD (Singular Value Decomposition) was our choice of algorithm as it gave the lowest RMSE score among all the models we tried. We tried different parameter settings to tune our model to achieve the best RMSE score and at the same avoid overfitting. eg:

algo = SVD(lr_all = 0.0035, n_epochs = 60, reg_all = 0.15, n_factors = 15)

We used the following four main hyperparameters to train the model.

- `n_factors` (number of factors)
- `n_epochs` (number of iteration of the Stochastic Gradient Descent procedure)
- `lr_all` (learning rate for all parameters)
- `reg_all` (regularization term for all parameters)

3 Result and Discussion

In this section we report and discuss about the results we have got from different algorithms and different sets of parameters. The results reported in Table 1 represents RMSE on the Kaggle dataset for different settings. As the number of maximum submission was limited, it was not possible to experiment with too many different settings. We summarize our insight from the results into the following points:

- We have inspected the performance with changing values of dimension (N) of the vector q_i and p_u from 50-200. We observed that increasing the value of N over 200 does not increase the performance significantly, which aligns with the findings of [3].
- Using ALS does not improve the performance for this experiment. So we decided to proceed with Stochastic gradient descent.
- Optimal value of λ is close to 0.07. Decreasing the value of λ after that does not improve RMSE significantly. Decreasing the value of λ beyond that may lead to overfitting by diminishing the regularization term.

- **Stochastic gradient descent with the bias term gives the best performance with an RMSE of 1.29482 on the Kaggle test set**.
- **Surprise package also performs very well with an RMSE of 1.29566 after tuning the parameters**.

References

[1] Y. Koren, R. Bell and C. Volinsky, 'Matrix Factorization Techniques for Recommender Systems', in Computer, vol. 42, no. 8, pp. 30-37, Aug. 2009.

[2] Surprise - A Python scikit for recommender systems: http://surpriselib.com/

[3] Yehuda Koren, 'Collaborative filtering with temporal dynamics', In Proc. of KDD 09.