System Report

Assignment 2:
Topic 3.4: Movies recommender system using collaborative filtering

Group: FunForFun

Ziyi Cui z5097491
Huijia Wang z5097494
Mingkai Zheng z5085507
Fengqing Liu z5095136

Introduction

With the development of social productivity and information based society, nowadays, people always face to variable selections of products both in reality and network. To take the advantages in business competition, the retailers that provide these choices need to enhance users experience and make users stay engaged with them. To achieve these effects, a key method is to offer the products that match the preferences of people as much as possible, which is the target of recommender system. Recommender system is a system that can roughly figure out the pattern of users' preferences and then recommend the products or information base on this pattern.

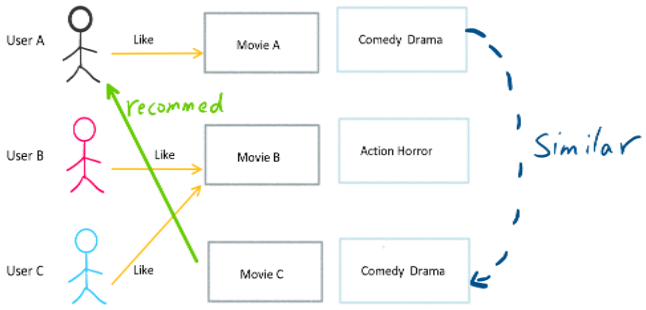
For instance, a user of a movie website want to watch a movie today, however, he doesn't know what movie to watch specifically since there are many movies on this website. In this situation, using recommender system can help this user find the movies that similar to the movies he had watched before and given high rating. As a result, this user may visit this movie website more frequently and the website achieve its goal of retain users.

This project aims to implement a simple movie recommender system that recommend some movies to a user that he may interest in, using collaborative filtering algorithm. The selection of movies will be based on the information of this user, such as gender, rating, age and so on.

Literature Survey

Since the topic is not covered in the lecturer, we did a lot of research before we start. There are two main methods to implement a recommender system, ***content-based filtering*** and ***collaborative filtering***.

Content-Based Filtering

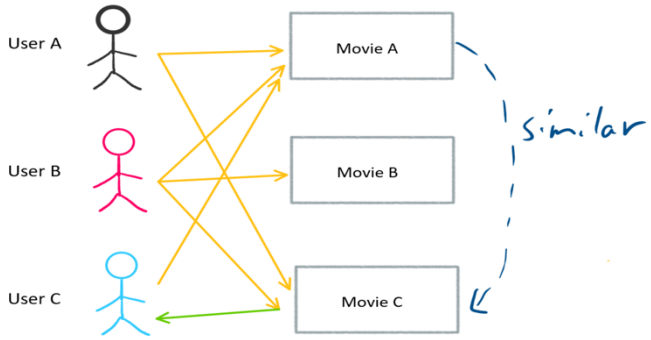


Just like the name of this method, the content-based filtering is based on content (the content is movie in this case). In Figure above, User A likes Movie A, the genres of Movie A are comedy and drama, there is another movie has the same genres which is Movie C. Hence, we can recommend Movie C to User A.
The method is user independent, when we want to recommend a new movie to a user, we don't need to grab some extra data from other users, all we need to consider is to check if the new movie has the same properties as the user's favourite movie.

Drawback of Content-Based Filtering

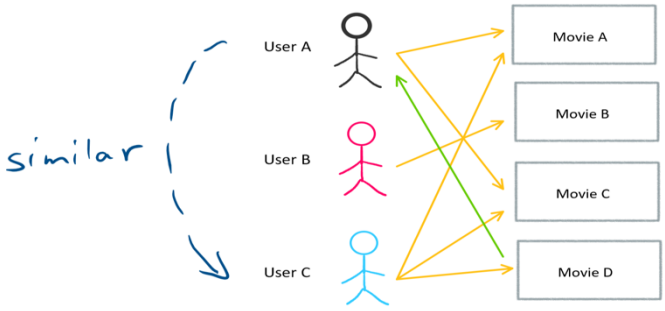
This method is simple and effective, but the drawbacks are also obvious. First, if a new user comes in to the system, we don't have any information about the user, so we don't know which movie the user watched before and what kinds movie the user likes, which means we have no reference to compare. Second, we need to prepare a large item profile for the system. Third, if a user just watched a comedy movie before, content-based filtering will only recommend a comedy movie to this user, so there is nothing "surprised" for a user.

User-Based Collaborative Filtering



User-based collaborative filtering will find the neighbours that have the same preference as the target user, then generate recommendations to target users based on the preference of the target user's neighbours.
In Figure 2, User A likes Movie A and C, User C likes Movie A, C, and D. From the history of these users, we can find that the tastes of user A and user C are similar, so we can recommend movie D to user A.

Item-based Collaborative Filtering



According to the evaluation of all users on the items, the similarity between the items will be found, then the similar item will be recommended to the user based on the user's historical preference information.
In Figure 3, User A likes Movie A and C, user B likes Movie A, B and C. we found that all users who like movie A also like movie C. Now, since user C likes Movie A, so we can recommend Movie C to user C.

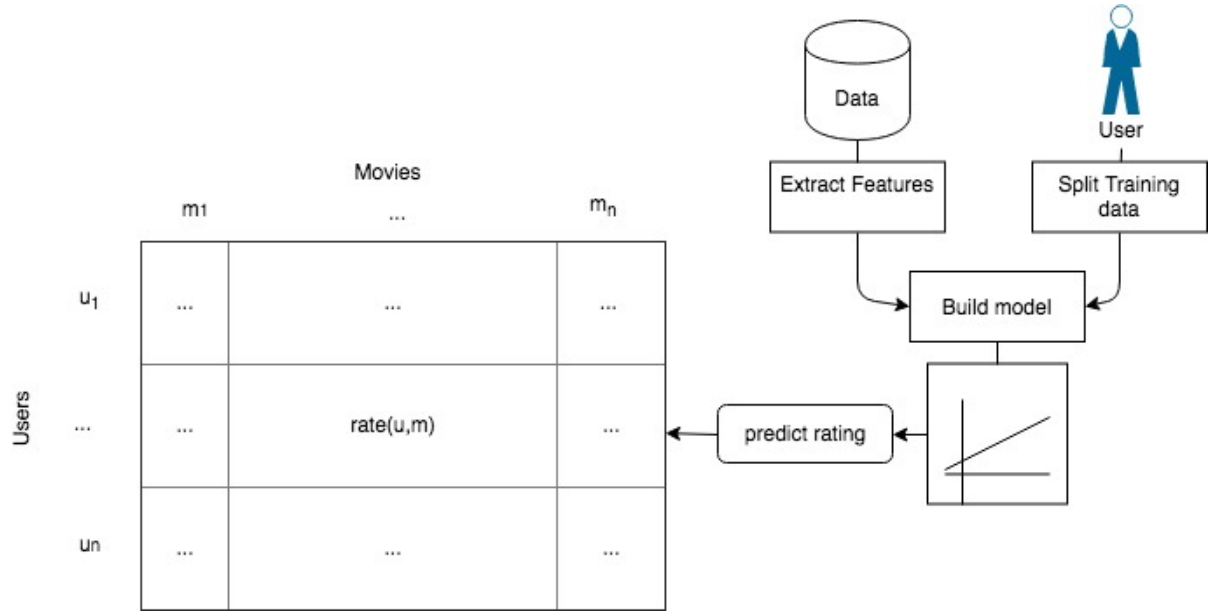
Drawback of Collaborative Filtering

The user based and item based collaborative filtering also have some drawbacks. The issue is spares data, if a user just watched one movie or he is a new user, there is a problem.

Matrix Factorization

This is the main method we used in our project, the key idea is to vectorize the movie features and user preference as vectors, we implement lots of different method based on matrix factorization, the details will be illustrated in the implementation section of this report.

Method Description



As the figure above, the way to construct a movie recommendation system can be divided to 4 main steps.

1. Extract features and split the training and test data(data preprocessing)
2. Build several training models and choose the one with the best preference
3. Predict the user rating base on the model we built
4. Give a list of unwatched movies based on the predicted rating from highest to lowest to a user

Data Preprocessing

In the original MovieLens data sets, we have 3 data files to describe the movies, users and rating respectively. In order to build the model conveniently, we merge these three data files into one file firstly. In this combined file, we have all features together: user id, movie id,

rating, timestamp, gender, age, occupation, zip, movie title and genres. And then, we extract user id, movie id and rating to build a new table for training. The table for training looks like below, where the columns represent the movie id and the rows represent the user id.

	001	…	i
001	Rating by user 001 on movie 1	…	Rating by user 001 on movie i
…	…	…	…
j	Rating by user j on movie 001	…	Rating by user j on movie i

Table 1: Rating by users on movies

And then, based on the above table, we create another table to indicate if user j rates the movie i. If user j rates movie i, we record an 1, otherwise we record a 0. This table looks like below:

	001	…	i
001	1	…	1
…	…	…	…
j	0	…	1

Table 2: Status of rating

Implementation

Matrix Factorization

In the data set, we have genres for each movie. According to the genre data of each movie, we say that each movie has some relevance with the genre they belongs to. Suppose that we transfer this relevance into a score and record each score into the genre columns this movie belongs to. Denote $x^{(i)}$ as the feature vector for each movie i, where the 1 is an interceptor term.

$$x^{(i)} = \begin{pmatrix} 1 \\ relevance\ score\ of\ genre\ a \\ \dots \\ relevance\ score\ of\ genre\ n \end{pmatrix}$$

Also, we can transfer the preference of each user to the particular genre, denote $\theta^{(j)}$ as the parameter vector for user j, where the 0 is an interceptor term.

$$\theta^{(j)} = \begin{pmatrix} 0 \\ preference\ to\ genre\ a \\ \dots \\ preference\ to\ genre\ n \end{pmatrix}$$

To recommend movies to a specific user, we can simplify this problem as predicting the score that a user will rate for a movie. Since if we can predict a user's rating of movies he has not watched, the movies with higher rating may be what this user interest in and then we can recommend these movies to that user. In order to do the prediction, we can treat predicting the rating of movies for each user into a separate linear regression problem. Therefore, for each user j and movie i, the predicted rating can be calculated as

$$\text{predicted rating} = \left(\theta^{(j)}\right)^T \left(x^{(i)}\right)$$

What we need to do now is to learn $\theta^{(j)}$, which is basically a linear regression problem. This means that we need to choose a $\theta^{(j)}$ so that the predicted rating is as close as possible to the values we observe from the original rating data. To write down the formulation, we need to define some notations first.

$Denote\ each\ cell\ in\ Table\ 1\ as\ y^{(i,j)}$

$$y^{(i,j)} = rating\ by\ user\ j\ on\ movie\ i$$

$Denote\ each\ cell\ in\ Table\ 2\ as\ r^{(i,j)}$

$$r^{(i,j)} = 1\ if\ user\ j\ has\ rated\ movie\ i\ (0\ otherwise)$$

$Denote\ the\ number\ of\ movies\ as\ m^{(j)}$

$$m^{(j)} = the\ number\ of movies\ rated\ by\ user\ j$$

The formulation to learn $\theta^{(j)}$ (the parameter vector for user j) will be:

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left[\left(\theta^{(j)}\right)^T \left(x^{(i)}\right) - y^{(i,j)}\right]^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(\theta_k^{(j)}\right)^2$$

Since we need to learn $\theta^{(j)}$ for each user in our data set, denote n_u as the number of users in our data set, the formulation to learn $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$ will be:

$$\min_{\theta^{(1)},\theta^{(2)},\dots,\theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left[\left(\theta^{(j)}\right)^T \left(x^{(i)}\right) - y^{(i,j)}\right]^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(\theta_k^{(j)}\right)^2 \qquad (1)$$

After we learn $\theta^{(j)}$, we can know the preferences of each user to each genre. Therefore, given $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$, we can learn $x^{(i)}$ by

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} \left[\left(\theta^{(j)} \right)^T \left(x^{(i)} \right) - y^{(i,j)} \right]^2 + \frac{\lambda}{2} \sum_{k=1}^{n} \left(x_k^{(i)} \right)^2$$

Since we need to learn $x^{(i)}$ for each movie in our data set, denote n_m as the number of movies in the data set, the formulation to learn $\mathrm{x}^{(1)}, \mathrm{x}^{(2)}, \dots, \mathrm{x}^{(\mathrm{n_m})}$ will be:

$$\min_{\mathrm{x}^{(1)}, \mathrm{x}^{(2)}, \dots, \mathrm{x}^{(\mathrm{n_m})}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left[\left(\theta^{(j)} \right)^T \left(x^{(i)} \right) - y^{(i,j)} \right]^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)} \right)^2 \qquad (2)$$

Following, we can apply collaborative filtering algorithm to minimizing $\mathrm{x}^{(1)}, \mathrm{x}^{(2)}, \dots, \mathrm{x}^{(\mathrm{n_m})}$ and $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(\mathrm{n_u})}$ simultaneously. Denote the optimization objective as $J(\mathrm{x}^{(1)}, \mathrm{x}^{(2)}, \dots, \mathrm{x}^{(\mathrm{n_m})}, \theta^{(1)}, \theta^{(2)}, \dots, \theta^{(\mathrm{n_u})})$, which is the combination of equation (1) and equation (2) above.

$$J\left(\mathrm{x}^{(1)}, \mathrm{x}^{(2)}, \dots, \mathrm{x}^{(\mathrm{n_m})}, \theta^{(1)}, \theta^{(2)}, \dots, \theta^{(\mathrm{n_u})}\right)$$
$$= \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left[\left(\theta^{(j)} \right)^T \left(x^{(i)} \right) - y^{(i,j)} \right]^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} \left(x_k^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} \left(\theta_k^{(j)} \right)^2$$

From this equation we can get the collaborative filtering optimization formulation.

$$\min_{\substack{x^{(1)}, x^{(2)}, \dots, x^{(n_m)} \\ \theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}}} J\left(x^{(1)}, x^{(2)}, \dots, x^{(n_m)}, \theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}\right) \qquad (3)$$

Normalization

As we mentioned in the section **Drawback of Content-Based Filtering,** the recommendation system is unable to recommend movies to the new user, who haven't rate any movies before. The strategy to solve this problem is recommend the movies with average rating from the highest to the lowest. In order to achieve that, we do normalization for our training rating matrix. In this method, we use all given ratings for each movie to subtract the average rating for the move in order to scalar the average rating for each movie to 0. After the normalization, the predict rating for new user will be the average rating to a certain movie.

Bias System

To improve the accuracy of the recommendation system, we introduce a bias system to represent the bias among different users and movies, which could effects the rating of movie to a certain user. For example, a better movie deserve a higher rating than an average level movie in general. And a critical person might give a lower rating to a movie than average users. Therefore, it's necessary to consider the bias in our recommendation system. We use b_u, b_m to represent the user and movie bias, and the overall rating denoted by μ. So, the bias can be showed like the equation below:

$$b = b_u + b_m + \mu$$

Combine the bias with the rating equation in Matrix Factorization:

$$r^{(u,m)} = b_m + b_u + \mu + \left(\theta^{(u)}\right)^T \left(x^{(m)}\right)$$

SVD++

To further improve the accuracy, we consider to involve implicit information for movies which basically shows how attractive which movie is – the more users choose it, the more attractive the movie is. In order to do so, we introduce two new value, imp_m the implicit feedback factor indicates the implicit information of movies and N(u) contains all movies rated by user u. Also in training process, we take imp_m into the caluclatioin of $x^{(m)}$ to adjust it. The user implicit feedback will be used to adjust the user rating factor. The user implicit feedback is represented by

$$|N(u)|^{-\frac{1}{2}} \sum_{n \in N(u)} imp_n$$

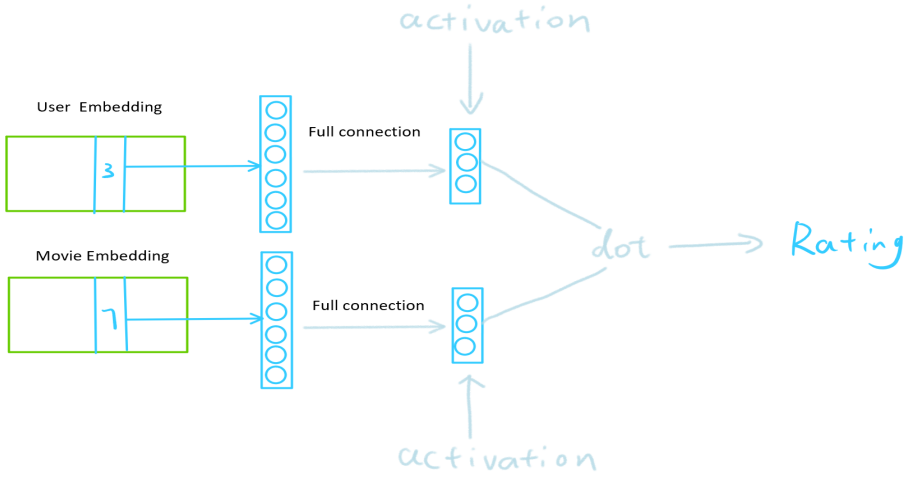
and the rating equation will be

$$r^{(u,m)} = b_m + b_u + \mu + \left(x^{(m)}\right)^T \left(\theta^{(u)} + |N(u)|^{-\frac{1}{2}} \sum_{n \in N(u)} imp_n\right)$$

Note that the implicit feedback will take larger place and the movie is rated by more users, so in larger datasets like movielens-20M SVD++ shows stronger performance.

Neural Networks with Embedding Layer

This is a neural network approach but the idea is pretty similar with the Matrix Factorization, and there is a little improvement. Initially, we embedding two matrices into our neural network, one for movie features, the other one for user preferences.
The different point is, in Matrix Factorization, we take the dot product of movie features vector and user preferences vector, the result will be the rating we predict in our model. But in the embedding approach, we take the movie features vector and user preference vector as the input of our neural network. The process has been shown below.



In this example, we get a training data -----> User 3, Movie 7, Rating 4.

we extract the vector 3 from user embedding layer and vector 7 from movie embedding layer, then feed them to the neural network, pass the activation function in next layer, finally we calculate dot product the output vector from the hidden layer, the result will be the rating we predict from the model.

Advantages
In this model, since we just feed the training data one by one to the neural network, so every time we just train one user vector and one movie vector. We don't need to worry about the empty rating issue in the original matrix factorization method.
The matrix factorization is a linear model, but we can simply use the hidden layer to transform the model to a nonlinear model.
Does not require a lot of pre-processing, because we don't need to construct a large rating matrix.

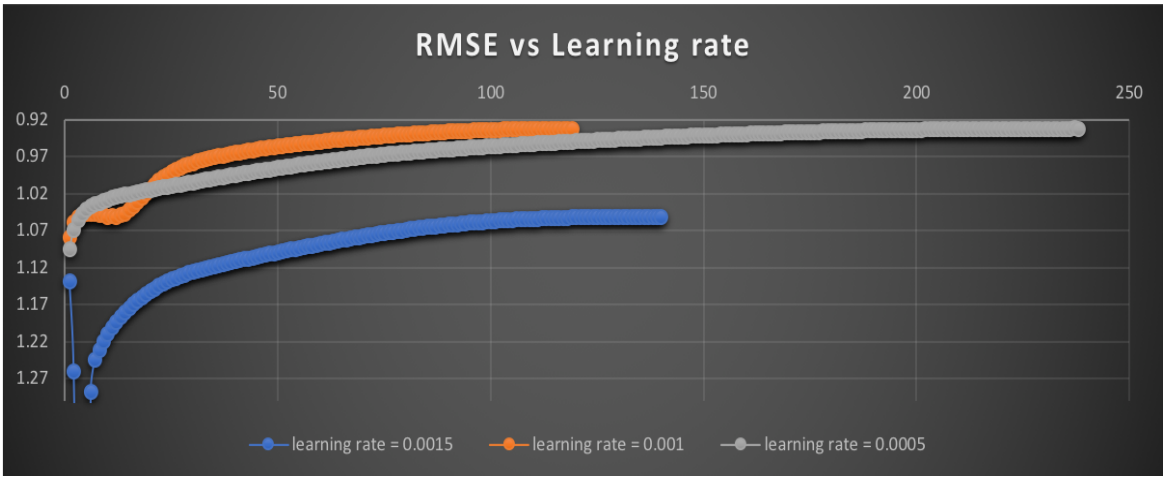
Disadvantages
Since we train the data one by one, so it takes very long time to train the model.
My code takes almost an hour to finish the training.

Recommend Movies

```
Cuisineeee@Zac-Cuis-MacBook ~/Documents/cs9417/project/recommender-system
master ● python3 recommender.py 1
1  Movie name: Contact (1997)
2  Movie name: Twelve Monkeys (1995)
3  Movie name: Leaving Las Vegas (1995)
4  Movie name: Blade Runner (1982)
5  Movie name: 2001: A Space Odyssey (1968)
6  Movie name: Brazil (1985)
7  Movie name: Mystery Science Theater 3000: The Movie (1996)
8  Movie name: Fifth Element, The (1997)
9  Movie name: Delicatessen (1991)
10  Movie name: Clockwork Orange, A (1971)
```

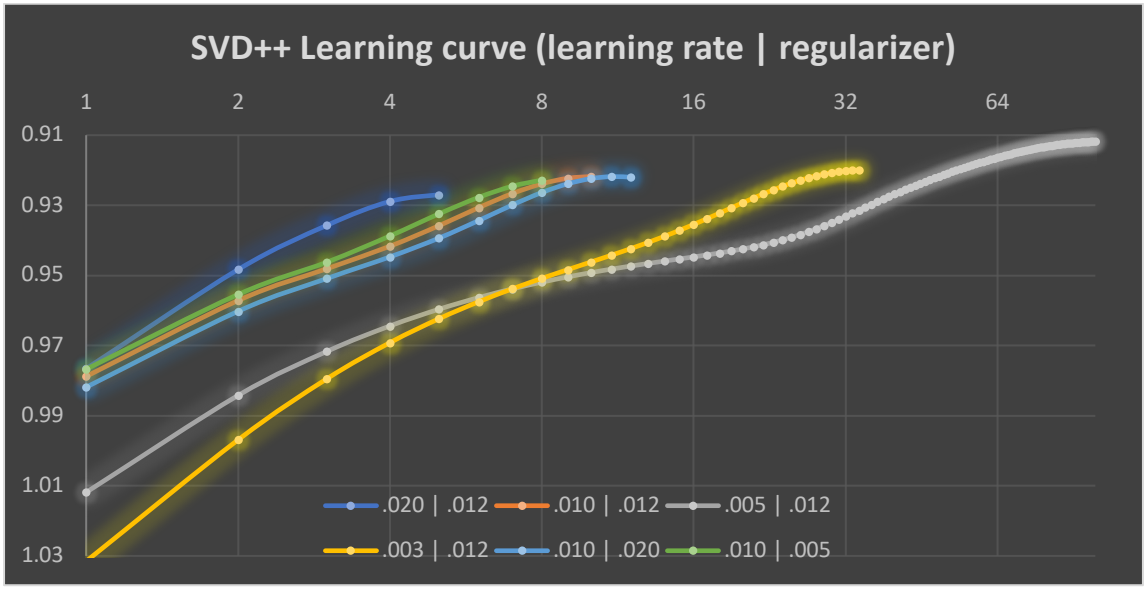
Performance Analysis

Matrix Factorization



The figure above show the learning curve of three different learning rates. When learning rate = 0.0015, it converges quickly but the final result is unsatisfactory compared with the others, which means that this learning rate is too rapid. Compared learning rate of 0.001 (orange line) and 0.0005 (grey line), it is clear too see that they reach the similar result where RMSE≈0.9315. However, when learning rate = 0.001, it reaches the end of learning more quickly. Therefore, we finally choose 0.001 as our learning rate of this method.

SVD++



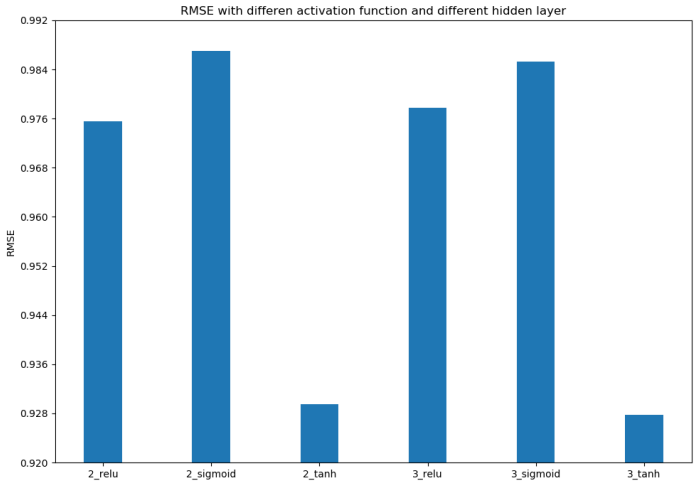
This figure shows the impact to learning curve by different learning rate and regularizer. It is logged on axis x to make a clearer image. By trying different combination of learning rate and regularizer we can see the combination of lr = 0.005 and reg = 0.12 gives a reasonable training time and most accurate result. Same as above, the higher the learning rate is, the quicker training speed but less accurate result you get. Also, the lower the regularizer is, the quicker but higher chance to overfit it could be.

Neural Networks with Embedding Layer

Since the number in the embedding matrices are random generated in normal distribution, if a new user comes in, the new user preference vector might be very odd, so we restrict the range of the number generation from -0.5 to 0.5 for both user and movie embedding matrices. We also subtract 3 from all the ratings, the range of rating scoring should have been between 1 and 5, but now it becomes -2 to 2. Now, if a new user and a new movie comes in to this system, the dot product of these two vectors should be around zero, which is the mean of -2 and 2. This is reasonable, because you don't know anything about the new user and the new movie, so we just assume the user will give the new movie an average mark.
I tried lots of different number of neural in hidden layer and different activation functions for this model.
Hidden layer neural = num_features/2 Activation function = tanh
Hidden layer neural = num_features/2 Activation function = relu
Hidden layer neural = num_features/2 Activation function = sigmoid
Hidden layer neural = num_features/3 Activation function = tanh
Hidden layer neural = num_features/3 Activation function = relu

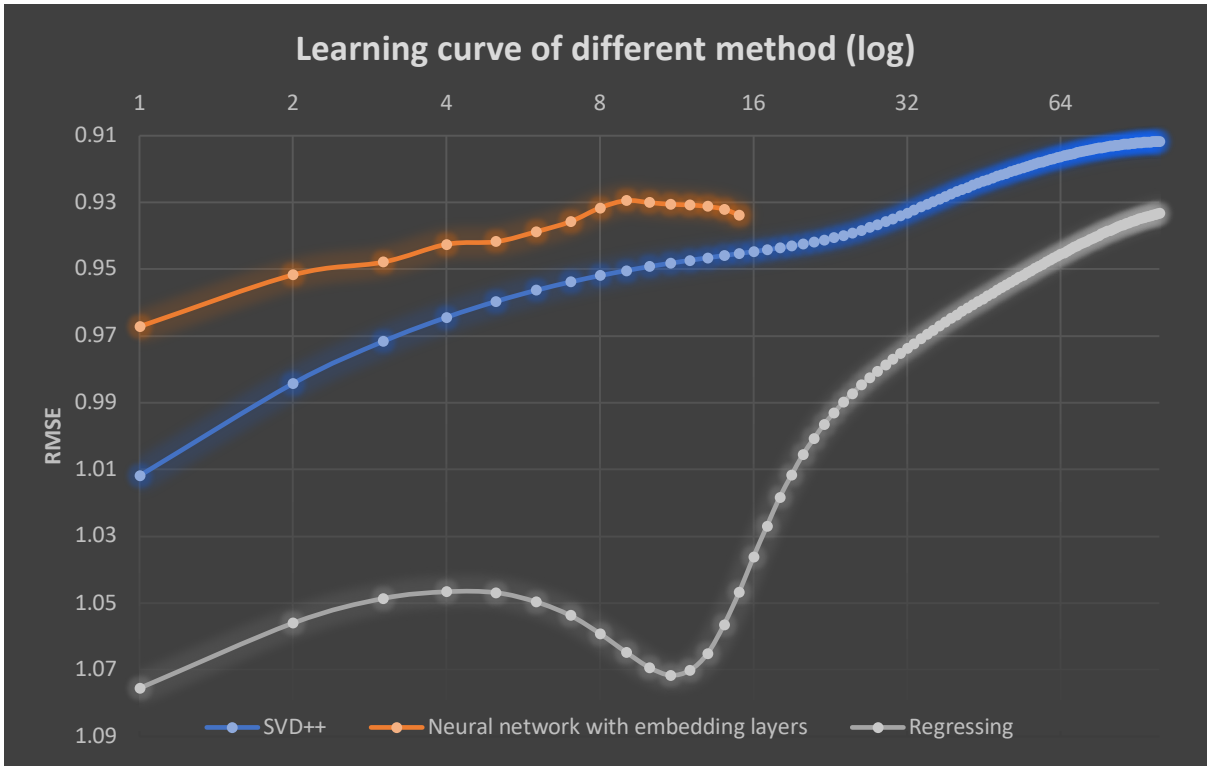
Hidden layer neural = num_features/3 Activation function = sigmoid

The performance has been shown below.



Hidden layer neural = num_features/3 Activation function = tanh has the best performance and we got RMSE≈0.927. The details of the experiments result are in the folder embedding_result.
The reason of this behaviour is that there is no negative output in sigmoid and relu.

Conclusion



The graph above shows the performance of each method we have used in this project. As we can see from the graph, the learning curve of regression is much more winding than the

others, but it reaches RMSE≈0.9332 at the end. The learning curve of neural network converges more quickly than the others but its RMSE cannot keep reducing at final stage and its smallest RMSE≈0.9279. The learning curve of SVD++ grows more gently and reaches RMSE≈0.9117.
In conclusion, since RMSE represents the error these methods occur during their learning, it is clear that SVD++ is the best method we can choose in this project as it has the smallest RMSE at the end of learning.

Relevance Test Results

The following figure shows some tests results from MyMediaLite website (http://www.mymedialite.net/examples/datasets.html). It is clear to see that our SVD++ result of RMSE≈0.9117 is not bad.

MovieLens 100k

5-fold crossvalidation with `--random-seed=1`

Method	--recommender-options	RMSE	MAE
BipolarSlopeOne		0.96754	0.74462
UserItemBaseline	reg_u=5 reg_i=2	0.94192	0.74503
SlopeOne		0.93978	0.74038
UserKNNCosine	k=40 reg_u=12 reg_i=1	0.937	0.737
UserKNNPearson	k=60 shrinkage=25 reg_u=12 reg_i=1	0.92971	0.72805
ItemKNNCosine	k=40 reg_u=12 reg_i=1	0.924	0.727
FactorWiseMatrixFactorization	num_factors=5 num_iter=5 shrinkage=150	0.9212	0.7252
BiasedMatrixFactorization	num_factors=5 bias_reg=0.1 reg_u=0.1 reg_i=0.1 learn_rate=0.07 num_iter=100 bold_driver=true	0.91678	0.72289
BiasedMatrixFactorization	num_factors=10 bias_reg=0.1 reg_u=0.1 reg_i=0.12 learn_rate=0.07 num_iter=100 bold_driver=true	0.91496	0.72209
SVDPlusPlus	num_factors=4 regularization=0.1 bias_reg=0.005 learn_rate=0.01 bias_learn_rate=0.007 num_iter=50	0.9138	0.71836
ItemKNNPearson	k=40 shrinkage=2500 reg_u=12 reg_i=1	0.91327	0.7144
BiasedMatrixFactorization	num_factors=40 bias_reg=0.1 reg_u=1.0 reg_i=1.2 learn_rate=0.07 num_iter=100 frequency_regularization=true bold_driver=true	0.90764	0.71722
BiasedMatrixFactorization	num_factors=80 bias_reg=0.003 reg_u=0.09 reg_i=0.1 learn_rate=0.07 num_iter=100 bold_driver=true	0.91153	0.72013
SVDPlusPlus	num_factors=10 regularization=0.1 bias_reg=0.005 learn_rate=0.01 bias_learn_rate=0.007 num_iter=50	0.91096	0.7152
BiasedMatrixFactorization	num_factors=320 bias_reg=0.007 reg_u=0.1 reg_i=0.1 learn_rate=0.07 num_iter=500 bold_driver=true	0.91073	0.72053
BiasedMatrixFactorization	num_factors=160 bias_reg=0.003 reg_u=0.08 reg_i=0.1 learn_rate=0.07 num_iter=100 bold_driver=true	0.91047	0.71944
SigmoidItemAsymmetricFactorModel	num_factors=5 regularization=0.005 bias_reg=0.1 learn_rate=0.006 bias_learn_rate=0.7 num_iter=65	0.91	0.71701
SVDPlusPlus	num_factors=4 regularization=1 bias_reg=0.05 learn_rate=0.01 bias_learn_rate=0.07 num_iter=50 frequency_regularization=true	0.90906	0.71547
SigmoidItemAsymmetricFactorModel	num_factors=10 regularization=0.005 bias_reg=0.1 learn_rate=0.006 bias_learn_rate=0.7 num_iter=90	0.9086	0.71522
SVDPlusPlus	num_factors=20 regularization=0.1 bias_reg=0.005 learn_rate=0.01 bias_learn_rate=0.007 num_iter=50	0.90829	0.713

Future Work

Because of the time limitation, we cannot implement more experiments to improve the performance of our recommender system. However, when we do the research for this project, we also find there are some methods which have higher RMSE and accuracy than the methods we have implemented above, such as Boltzmann Machine and Autoencoders.

These methods can build a more robust recommender system and they are important for us to make an improvement in the future.

References

Vig, J., Sen, S., and Riedl, J. 2012. The tag genome: Encoding community knowledge to support novel interaction. ACM Trans. Interact. Intell. Syst. 2, 3, Article 13 (September 2012), 44 pages. DOI = 10.1145/2362394.2362395 http://doi.acm.org/10.1145/2362394.2362395

Koren, Y., Bell, R. and Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. Computer, 42(8), pp.30-37.

Sarwar, Badrul M, George Karypis, Joseph A Konstan, and John T Riedl. 2000. “Application of Dimensionality Reduction in Recommender System – A Case Study.” In WebKDD 2000. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.8381.