Advances in Data Mining

Assignment 1 - Recommender System

Sevak Mardirosian
s2077086
s.mardirosian@umail.leidenuniv.nl

Georgios Kyziridis
s2077981
g.kyziridis@umail.leidenuniv.nl

September 25, 2017

Abstract

Nowadays, recommender systems are used to personalize your experience on the web, telling you what to buy, where to eat or even who you should be friends with. People's tastes vary, but generally follow patterns. People tend to like things that are similar to other things they like, and they tend to have similar taste as other people they are close with. Recommender systems try to capture these patterns to help predict what else you might like. E-commerce, social media, video and on-line news platforms have been actively deploying their own recommender systems to help their customers to choose products more efficiently, which serves win-win strategy. In this paper we will display our approach and what techniques we used to implement a simple recommender systems using four naive approaches, Matrix Factorization Gradient Descent given a dataset from dataset MovieLens.

1 INTRODUCTION

A Recommender System predicts the likelihood that a user would prefer an item. Based on previous user interaction with the data source that the system takes the information from (besides the data from other users, or historical trends), the system is capable of recommending an item to a user. Think about the fact that Amazon recommends you books that they think you could like; Amazon might be making effective use of a Recommender System behind the curtains. This simple definition, allows us to think in a diverse set of applications where Recommender Systems might be useful. Applications such as documents, movies, music, romantic partners, or who to follow on Twitter, are pervasive and widely known in the world of Information Retrieval.

The problem we are addressing at this point is to simulate a recommender system in order to make life easier for some of us on-line. They are by many different methods and techniques to implement a proper Recommender system. Two most ubiquitous types of recommender systems are Content-Based and Collaborative Filtering (CF). Collaborative filtering produces recommendations based on the knowledge of user's attitude to items, that is it uses the "wisdom of the crowd" to recommend items. In contrast, content-based recommender systems focus on the attributes of the items and give you recommendations based on the similarity between them.

Upon mentioning recommendation systems we might find ourselves struggling with two different concepts: One is called prediction, it consists of finding a numerical value which express predicted likeliness of an item for a particular user, or vice-versa. And the second one called recommendation. It basically means finding a list of items that the user might find this interesting as well based on previous items.

On the other hand matrix factorization methods are one of the most successful realization of the latent factor models. They are superior to the classic nearest-neighbor techniques under different aspects. They provide scalability, prediction accuracy, flexibility. They rely on different type of input data.

1.1 Organization

The paper is structured as follows: In Section 2 we state a formal definition of the recommendation problem, in Section 3 we describe the algorithms we implemented and studied, in Section 4 we describe the dataset used, and we present our results and their interpretation. In Section 5 we give some conclusions and propose some future work.

2 PROBLEM STATMENT

In recommendation system there are 2 classes of entities, users and items. Users have different preferences for variety of items. List of n users $U = u_1, u_2, ..., u_n$ and a list of m items $I = i_1, i_2, ..., i_m$. Let $R = R_{ui}$ for $u = 1...n$ & $i = 1...m$ denote the user-item matrix where R_{ui} represent the rating score of item i rated by user u. Some of the items have ratings others do not (basically 0). So the goal of this paper is to build a recommender system to estimate the missing values in R based on the already known values. The data is stored as a utility matrix which gives, for every pair user-item, a value that represents what is known about the degree of preference of that user for that item. The matrix is usually sparse; that means that most of the values are missing. Normally the goal is not to predict every blank entry, but rather to determine some entries in each row which is likely to be high. This means, we usually focus on predicting high-valued ratings. We refer to a prediction of a rating from user u of the item i with the hat superscript notation, that is, $\hat{R}_{ui}$ is the prediction of R_{ui}.

So the goal of the recommender systems ends up to find an approximation of the rating matrix R, which we refer to as $\hat{R}$, such that the prediction error is minimized.

3 ALGORITHMS

We implemented several algorithms in this experiment in order to test their results and evaluate their differences as in the development part as in the efficiency dissimilarity.

3.1 Naive Approaches

These approaches seem to be surprisingly well. They are based on the idea that the user will usually have an average behavior,and they return an average value as the predicted value. Below the explanation of each of those models.

- Global Average Approach

  The first Naiv approach was to calculate the global average value from the hole column 'Rating' which includes marks of a movie from a user. In order to test the prediction error we implemented the RMSE and then we calculated the average value for the errors in the trainset and in the testset. The algorithm is based on a general aspect of the total average for the movie ratings, so it is not using much useful prior information in order to generate significant predictions.

  $$\text{Global Average} = \frac{1}{N}\sum_{i=1}^{n} ratings_i$$

- User Average Approach

  The Second approach was to calculate the average rating for each user, so to experiment with a prediction-recommendation based on the unique preferences of each user. The idea behind this approach is that the recommender system's suggestions are generated by using prior information based on the user's choices and that fact has a strong impact on the prediction error. The mean values of each user were the predicted values on which we implement the RMSE. As result, suggestions would be closer to the user general habits and the recommended movies will fit in their own aesthetic quality. The drawback in this algorithm is that the system recommends without using information from the other users so maybe has limited power because of lack in more informative prior.

  $UserAverage_i = \sum_{k=1}^{K_i} \frac{1}{K_i}\, ratings_{ik}$, for $i = 1, 2, 3...., I$: user_id and $k = 1, 2, 3..., K_i$: ratings.

- Item Average Approach

  The third approach is exactly the same with the above and it is based on average of each movie. So first we calculate the mean value for each movie from the column of 'rating' and then, we test again the calculated errors on the same procedure. In this case the positive facts are that the recommender system suggests based on the similarities of movies. If there is a blockbuster movie inside the dataset then the high ratings does not appear to be unreasonable. The drawback here is that the system provides predictions and suggestions which are popular and they do not take into consideration the user's preferences.

  $MovieAverage_i = \sum_{k=1}^{K_i} \frac{1}{K_i}\, ratings_{ik}$, for $i = 1, 2, 3...., I$:movie_id and $k = 1, 2, 3..., K_i$:ratings.

- User-Item Linear Regression

  The last Naiv approach we implemented is called Linear Regression. This is an algorithm which produces a mathematical model that generate predictions using the vector of means for each user and the vector of means for each movie. That model can visualized by a line. The specific model in this case is : $R_{user,item} = \alpha R_{user} + \beta R_{movie} + \gamma$. The algorithm, using as prior information the two vectors R_{user} and R_{movie}, generates coefficients (α, β, γ) calculated

by distance methods which are estimating the minimum error between the values of the two vectors. As result, the algorithm produces a formula which provides information about the relation of our data by estimating a line which approximately fits in the observations.

3.2 Matrix Factorization - Gradient Descent

- Matrix Factorization

  The idea behind $MatrixFactorization$ is to approximate a matrix as a product of two other matrices. Let A be a $m \times n$ matrix of rank r. The theorem says that if there is a $m \times r$ matrix X and a $r \times n$ matrix Y, we can express matrix A as the product of the two matrices like : $A = XY$. So X consist of r columns and Y r number of rows. The interesting feature in this approach is that we can split matrices and compress them with low rank. More precisely, concerning the storage of matrix A we have to input m times n values while for matrix X the input is m times r and for Y is r times n values. So for A we have mn numbers and for the X and Y we have $mr+rn$ numbers, then : $mn > r(m+n)$. For instance,

  $A = \begin{bmatrix} 1 & 2 & 3 & 5 \\ 2 & 4 & 8 & 12 \\ 3 & 6 & 7 & 13 \end{bmatrix}$, we can produce linear equations : $\begin{matrix} A_1 = 1A_1 + 0A_3 \\ A_2 = 2A_1 + 0A_3 \\ A_3 = 0A_1 + 1A_3 \\ A_4 = 2A_1 + 1A_3 \end{matrix}$, for $A_1 ... A_4$ the columns of matrix A. So we can assign $X = \begin{bmatrix} 1 & 3 \\ 2 & 8 \\ 3 & 7 \end{bmatrix}$ and $Y = \begin{bmatrix} 1 & 2 & 0 & 2 \\ 0 & 0 & 1 & 1 \end{bmatrix}$, so the product of these two matrices is $XY = \begin{bmatrix} 1 & 2 & 3 & 5 \\ 2 & 4 & 8 & 12 \\ 3 & 6 & 7 & 13 \end{bmatrix} = A$

- Gradient Descent

  This algorithm is a way to denote the minimum or the maximum value of a mathematical function. It is very useful in the field of machine-learning and data-mining because it is an efficient method of estimating optimal values using a sequence of derivatives. It basically used to estimate optimal wights using calculating derivatives with a learning rate (λ).

  Example :

  Assume we have a $f(x) = x^2 - 2x + 2$, for $x \in \mathbf{R}$. We know that the minimum value of x is 1. With gradient descent we can develop a formula like : $X_{i+1} = X_i - \lambda f'(x)$, for λ the rate of learning(step). So first we have to define the $f'(x) = 2x + 2$ derivative, pick a random number for X, $x = 3$ and then use it in the above formula with $\lambda = 0.2$. The equation is like :

  $$\begin{matrix} f'(x) = 2x - 2 \\ X_1 = x_0 - 0.2f'(3) \Rightarrow \\ X_1 = 3 - 0.2(4) \Rightarrow \\ X_1 = 2.2 \end{matrix}$$

  $$\begin{matrix} X_2 = 2.2 - 0.2f'(2.2) \Rightarrow \\ X_2 = 2.2 - 0.2(2.4) \Rightarrow \\ X_2 = 1.72 \end{matrix}$$

  Repeat this process until find the minimum...

4 EXPERIMENTS

The procedure of the implementation of the above algorithms is developed in python using the dataset MovieLens. In order to test our results we split the initial dataset into two smaller sets using Cross-Validation.

4.1 Dataset

The datasets we experimented with was instructed to fetch from MoviesLens (1M) collected by GroupLens Research from the MovieLens web site.The dataset contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users. For this purpose of the assignment we only and mainly used the ratings.dat file to experiment with. Given the different matrix values no clean-up and and normalization was required to load the data faster and get rid of the missing movies-ratings. The dataSet contains three columns : User_id | Movie_id | Rating |.

4.2 Cross-Validation

In this experiment we have implemented a 5-fold cross-validation which means that the operation of splitting the datasets and estimating the predictions, implemented five times. For each time(fold) we produced predictions using the TrainSet observations, calculated the error values and compared them with the error values of TestSet. The remarkable fact in this algorithm is that for each time(fold) in cross-validation process the TrainSet and the TestSet fulfilled with different observations randomly. That was the way to evaluate the differences between the error values. The error function we used to calculate that value is the Root Mean Square Error (RMSE). This is a formula which implements the sum of the squared differences between predicted and ground truth values, fragmented by the number of all observations. That is the reason of its name, it is the average value of the squared Δ between the prediction and the truth value.

$$\text{Root Mean Square Error} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(predicted - true)^2}$$

Further we run our experiments using Python/Numpy suite on dual-core laptop Intel i7 type CPU running with 8GB of RAM and a total of 4 cores.

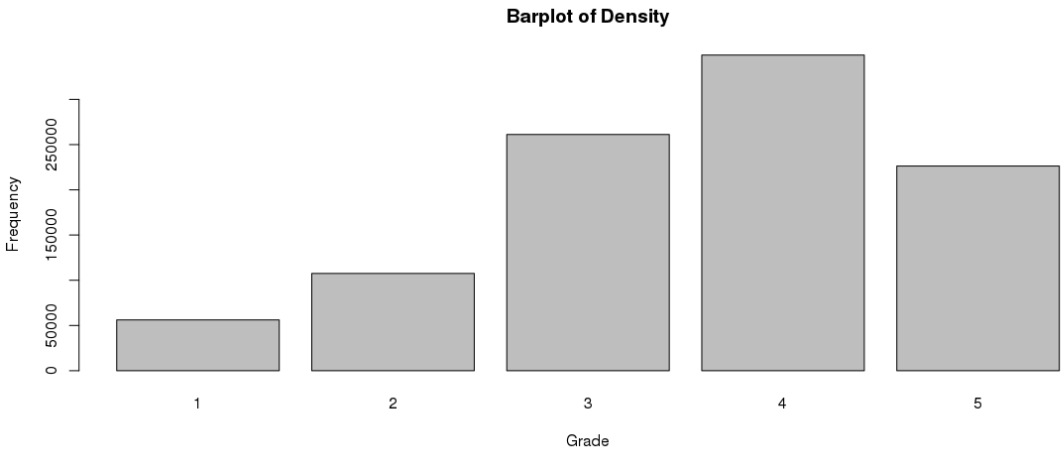
4.3 Results

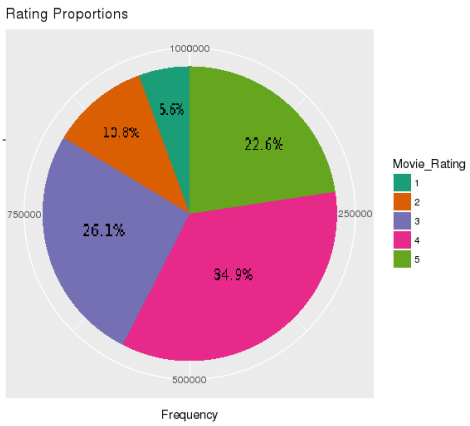
In this section we report the results of our experiments on the dataset, and then we analyze our findings. Every reported measure is the average of the 5 runs of the 5-fold-validation technique. The results from the table below provides information about the following: All algorithms were implemented, Root Mean Square Error, Mean Absolute Error, the machine memory and the implementation time.

We started started with the traditional naive approaches and afterwards implmenting matrix factorization. In GD we were recommended to use the following params: numfactors=10, numiter=75, regularization=0.05, learnrate=0.005

Algorithm	RMSE	MAE	Memory(mb)	Time (sec)
Global avg	1.117	0.9339	421	0.627
User avg	1.035	0.8305	421	0.795
Item avg	1.202	1.0836	421	0.791
User-Item avg	1.116	0.762	420	827
MF-GD	0.8207	0.6894	422	5161

The two charts below, provide information about the frequencies of the ratings that the users have voted. Each column of the Barplot and each slice of the pie visualize the proportion of the frequency for each rank. The ratings for the movies are between 1 and 5 and it can be easily seen that the maximum proportion is on the grade 4 counted by 34.9%. On the contrary, the minimum proportion, as expected , is on the grade 1 with 5.6%.





4.3.1 Execution time

The execution time measured in seconds for every algorithm, in log scale, for a single step of the 5-fold-validation. Naive approaches are the fastest,Matrix factorization techniques are slower with

a running time on the order of magnitude of hours. In general, the GD algorithm execution time grows linearly with the number of ratings. If we would consider to update every single value in U or M as the basic operation for the complexity we then would have the time required for a single step in GD is about $O(nrnf)$. So the time grows linearly along with the number of ratings and number of items.

4.3.2 Maximum memory occupation

We calculated the memory usage using a resource module in Python, measured in Megabytes for every algorithm. The naive approaches have pretty much similar outputs. The GD algorithm has, also, the same memory requirements the naive approaches. The the memory required by the algorithms grows with the number of users and movies, for every algorithm.

5 CONCLUSIONS

5.1 Algorithm Conclusions

In this paper, we used the MovieLens 1M dataset to benchmark four different recommender systems. We tried to measure the execution time, the accuracy, and the memory requirements,understanding the main characteristics of every approach. One of our findings that we noticed that there is a consistent ratio between runtime and error measures, meaning, the lower the runtime the higher the error.

In terms of scalability, so far we believe that Matrix Factorization shows most promising results unless ALS (which we didn't have the time to implement) shows or proves to be more promising scalable algorithm.

It is obvious that the biggest error occurs in the first and the third Naive Approach Global Average and Movie Average. That means that the specific approaches do not provide significant predictions.That fact has strong impact on the recommender system because it produces predictions and suggests items which may are not reliable according to the user's desires. The basic drawback in these aspects is that the prediction is produced by non-significant prior information from the dataset so the error value is big.

The lowest error value occurs in the last algorithm called "Matrix-Factorization", that means that, this algorithm is more efficient because it generates predictions which are more reliable. The fact that this algorithm uses procedure of finding near-neighbors(similarities) between the users and the movies, provides very significant results. The idea behind the similarities between the objects of this experiment, is to expand and upgrade the prior information input of the algorithm using observations from similar users or movies and generate predictions according not only from the user preferences individually, but from a wide spectrum of users as well. The disadvantages of Matrix-Factorization is that it binds big amount of system's memory and consumes very big amount of processing power. Moreover, individual characteristics of each specific user are not taken into account so the system is tend to recommend popular items.

Future work could include implementing and benchmarking more recommendation algorithms such as collaborative filtering and content based recommendations. Other directions could be implementing other different factorization algorithms, and running more tests to get more consistent averages over the folds of the k-fold-validation technique.

5.2 General Conclusion

According to the table in the *Results* we can easily come to a conclusion that the most efficient algorithm in this experiment is the $Matrix - Factorization$ because of it's optimal errors. Nonetheless, its weakness of suggesting basically popular items would be probably an issue for some users. However,in respect of the processing power usage, the $NaivApproaches$ because of

it's limited system costs are more preferable instead of $Matrix - Factorization$.

References

Data downloaded from this website : http://grouplens.org/datasets/movielens/
Paper for Recommender Systems : gravity-Tikk.pdf