Recommender systems

Advances in Data Mining. Assignment 1

Abstract

Recommender systems are nowadays widely used to predict customers needs. We implement five different models for recommender systems: global average, movie average, user average, linear regression of the movie and user average and matrix factorization. Evaluation of the models is performed using the 1M MovieLens data set, using five-fold cross validation. We find matrix factorization to result in the best predictions and the global average to result in the worst predictions. Additionally, we measured the computational times and memory needed for all models. Based upon this, we recommend using linear regression for a recommender system where computational time is important and, if not, matrix factorization.

1. Introduction

Recommender systems are information filtering systems that try to anticipate the user by predicting it needs. Nowadays, these kind of systems are extensively used by almost all major web-based companies. Youtube tells the user which videos he probably wants to see next, Amazon which products he wants to buy and Spotify to what music he wants to listen.

The increasing amount of information that need to be processed has obliged the systems to evolve during the last decades, resulting in various different recommender methods. Originally, the nearest-neighbor technique was used, where the user is compared to other users to find those with the same taste. Items liked by them are then recommended to the user since he will probably like it too [4]. Later, Netflix, an online DVD-rental and video streaming service, promised a reward of 1 million dollars to the team that could improve their recommender system, Cinematch, by 10%. The winning team made use of matrix factorizations instead of the classic nearest-neighbor

techniques [3], proving it a better method than the nearest-neighbor technique.

In this report, we apply different recommender systems to a database containing the ratings that users of a website gave to a list of movies. The performance and results of the systems tested are presented and discussed.

2. The data set

The *MovieLens 1M* data set[1] is used in this work. This data set has been collected by GroupLens Research from the MovieLens website [2]. This data set consists of a set of $N = 1,000,209$ anonymous ratings $\{R_n\}_{n=1}^{N}$. Every rating R_n has four components: a user ID u_n, a movie ID m_n, a rating r_n on a 5-star scale (whole-star ratings only) and a timestamp t_n, in other words $R_n = (u_n, m_n, r_n, t_n)$. In this work only the first three components are used. In total there are 6040 distinct users, 3952 distinct movies and every user has rated at least 20 movies. For a more complete overview of the data set the reader is referred to its source.

3. Methods

3.1. Recommender systems

The goal of this research is to evaluate the performance of different implementations of a recommender system. A recommender system applies knowledge from the data set as represented by a *model* to create *predictions* for missing entries which can be ranked and used to make *recommendations*. In order to achieve this it is useful to order the data set in a specific way using a utility matrix U. The utility matrix consists of columns representing the movie IDs and rows representing the user IDs. Every entry $r_{ij} = (u_i, m_j)$ contains the rating corresponding to user u_i and movie m_j. Not all users rated all movies and therefore many entries are missing, therefore the utility matrix is sparse. A model can be thought of as a function of the utility matrix $M(U)$ which maps upon a prediction matrix $P = M(U)$. The prediction matrix contains predictions $p_{ij} = (u_i, m_j)$ for every user and movie.

[1] https://grouplens.org/datasets/movielens/1m/

High predictions suggest a certain user might like a certain movie and can be recommended in the case the user has not rated this movie already.
In the next Subsection we will introduce five different models used. After this Subsection we will describe how we evaluated these different models to quantify their performance.

3.2. Models

3.2.1. Global average

The simplest approach to predict how a user would rate a new movie is the global average of all ratings. The global average is defined as

$$p_{ij} = \frac{1}{N} \sum_{i,j} U_{ij} \tag{1}$$

where N indicated the total number of non-empty entries within the utility matrix.

3.2.2. User average

One could hypothesize that a critical user is more likely to have a low average movie score than a complimentary user. Therefore, a user average defined as

$$p_{ij} = \frac{1}{N} \sum_{j} U_{ij} \tag{2}$$

can be used to generate predictions. In case no user average can be computed, the global average is used instead.

3.2.3. Movie average

Similar to the user average, one can expect some movies are labeled as 'bad' and others as 'good' by many users. Therefore, a movie average defined as

$$p_{ij} = \frac{1}{N} \sum_{i} U_{ij} \tag{3}$$

can be used to generate predictions as well. In case no movie average can be computed, the global average is used instead.

3.2.4. Linear regression

Ideally, one would want to combine both the information and the movie to make a prediction. The simplest approach to this problem assumes a linear combination of the user and movie average:

$$p_{ij} = \alpha p_{ij,useravg} + \beta p_{ij,movieavg} + \gamma. \tag{4}$$

This a multiple linear regression problem. We used the Python package *statsmodels* to solve this equation for α, β and γ. In case of an empty ranking, the global average is used instead.

3.2.5. Matrix factorization

Matrix factorization (or incremental SVD) assumes every movie or user can be characterized by a certain number of factors n_{factors}. One could think of these factors as describing features of the movie and the user: for example the first movie factor may describe how much action a movie has and the corresponding first user factor how much the user likes action movies. If we order these factors in user and movie vectors, then large dot products correspond to matching movies and users. Since we have to calculate this dot product for all user and movie vector combination it can be described by matrix multiplication

$$X = UM, \tag{5}$$

where X is the matrix containing the ratings, M a $n_{\text{factors}} \times n_{\text{movieIDs}}$ matrix and U a $n_{\text{userIDs}} \times n_{\text{factors}}$ matrix.
In order to find the best U and M describing the data set we use a method similar to [1]. The method can be described as followed:

1. First we initialize the matrices U and M randomly. The random seed used to fill the matrices U and M before the matrix factorization was 2. Then, the matrices were filled with random values out of a normal distribution with the center at zero and a standard deviation of 0.1.
2. Now we fix all values of U and M except for one row of U and one row of M. Now we want to alter the entries in such a way that the difference between X and UM becomes smaller. Therefore, we calculate the error between the prediction p_{ij} and the observed rating x_{ij}, $\epsilon_{ij}^2 = (x_{ij} - p_{ij})^2$ where $p_{ij} = \sum_{k=1}^{10} u_{ik} m_{kj}$. Since we want to minimize this error, we want to update the values of U and M in the direction that minimizes the error. Therefore, the partial derivatives of the errors are calculated:

$\frac{\partial \epsilon_{ij}^2}{\partial u_{ik}} = -2\epsilon_{ij} m_{kj}$ and $\frac{\partial \epsilon_{ij}^2}{\partial m_{kj}} = -2\epsilon_{ij} u_{ik}$. The new values are updated in these directions with a certain step size, the learning rate η, which we set equal to 0.005. Furthermore to prevent over fitting we use a regularization λ which we set equal to 0.05. Summarized, we update the values as followed

$$u'_{ik} = u_{ik} + \eta(2\epsilon_{ij} m_{kj} - \lambda u_{ik}) \tag{6}$$

$$m'_{kj} = m_{kj} + \eta(2\epsilon_{ij} u_{ik} - \lambda m_{kj}). \tag{7}$$

This process is iterated 75 times for the full matrices U and M. In the end the prediction matrix equals the final UM.

3.3. Evaluation of the models

In the real world we want to implement the model that has the best predictions. In order to evaluate the models we will use 5-fold-cross validation. This means we randomly split the data set into five parts. The random seed used to randomly shuffle the entries of the initial data set (before it was split in 5) was 1. Four parts are combined to a training set, used to create predictions. One part will make up the test set, used to evaluate the model against. For the evaluation we use two statistics: the Root Mean Square Error (RMSE) defined as

$$RMSE = \frac{1}{N} \sum_{i=1}^{N} (rating - prediction)^2 \tag{8}$$

and the Mean Absolute Error (MAE) defined as

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |rating - prediction|. \tag{9}$$

The lower both statistics, the better the predictions of the model. Since we split the data into five parts there are five different combinations of training sets and test sets. For all combinations we calculated the RMSE and MAE. The final RMSE and MAE used are the averages of all five.

System parameters

All the codes and data reduction where performed in the same computer with characteristics as summarized in Table 1.

Architecture	x86-64
Operating System	Fedora 25 Twenty Five
CPU op-mode(s)	32-bit, 64-bit
CPU(s)	4
Model name	Intel(R) Core(TM) i5-6500 CPU 3.20GHz
Memory	7.7 GiB
Hard drives	463.7 GB

Table 1: Our system parameters.

4. Results

The MAE and the RMSE of the different recommender methods can be found in Table 2. The results are ordered with decreasing MAE and RMSE. The best parameters found for the linear regression model are: $\alpha = 0.78, \beta = 0.86, \gamma = -2.35$. The global average makes the worst predictions and the incremental SVD the best.

Recommender system	MAE	RMSE	time [s]	memory
Global average	0.93	1.12	0.39	$\mathcal{O}(1)$
Movie average	0.83	1.04	0.42	$\mathcal{O}(M)$
User average	0.78	0.98	0.56	$\mathcal{O}(U)$
Linear regression	0.73	0.92	1.67	$\mathcal{O}(R)$
Incremental SVD	0.68	0.88	1733.47	$\mathcal{O}(UMK)$

Table 2: This table shows the MAE and the RMSE found for every recommender system tested as well as the time needed to compute it and the memory used. As expected, for the naive approaches the linear regression is the most accurate but the incremental SVD technique is even better.

Additionally, we also include the time and memory ($\mathcal{O}$) needed to compute the prediction matrix. The computational time for all methods are of the same order, except for the incremental SVD, which takes about two order of magnitudes longer to compute, although this does also depend on the total number of iterations performed. For the global average the memory storage does not depend on the size of the data set, since we only need to store a cumulative counter and the number of items counted so far. For the movie and user average we need to store M movie averages and U user averages so

memory scales with the number of users and movies respectively. For linear regression we need to evaluate a prediction for every non-zero rating in the utility matrix, thus this scales directly with the total ratings. Lastly, for the incremental SVD for every movie and user combination (UM combinations) we need to evaluate the dot product of two vectors with length K, thus the memory needed equals UMK.

5. Discussion

As expected, the global average has the worst MAE and RMSE but needs the least amount of time and memory. Linear regression takes about the same time to compute as the global, movie and user average. Therefore, we would recommend using a linear regression recommender system if computational times need to be short.
Furthermore, the user average performs slightly better than the movie average. This might suggest the preferences of a certain user are more important than the movie characteristics. However, in this data set, it could also be explained by the fact that only users who rated many movies are included, therefore there is more information available for the user than for a specific movie.
Using linear regression improves the RMSE and MAE compared to the recommender systems discussed above. The required memory scales with the number of ratings (R), therefore, the time in order to compute it increases with more than one second.
The incremental SVD system introduces matrix factorization. Again, this improves even further the resulting statistics at cost of a huge increase in memory and computation time needed. The repeating iterations can, of course, be changed to reduce the time, but this will also reflect back on the results.
If we compare our results with the ones published in `http://mymedialite.net/examples/datasets.html` we can see that they are practically identical, adding confidence to our implementations of the algorithms.

6. Conclusions

In this work we analyzed the 1M MovieLens data set consisting of five-star-scale ratings given by users to movies. Using this data set, we evaluated different models for making predictions that can be used in a recommender system. The main conclusions of this analysis are:

- The best predictions are made using incremental SVD. Linear regression makes the second best predictions, followed by respectively, the user average and movie average. As expected, the simplest and most naive approach, namely the global average, had the worst predictions.
- We recommend linear regression as the best model if computational times are important. This model has the best balance between low MAE and RMSE and computational time.
- We recommend incremental SVD as the best model if computational times are not important.

References

[1] Ta Acs, Bott An Emeth, and Domonkos Tikk. On the Gravity Recommendation System.

[2] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015.

[3] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.

[4] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.