Advances in Data Mining: Assignment #1

Anonymous

Leiden University — October 3, 2020

Introduction: Recommender Systems

In the last years, many e-commerce and retail companies have been leveraging the power of data and boost sales by implementing recommender systems on their websites. These systems aim to predict user's interests and recommend items that are quite likely to be interesting for them. Data required for recommender systems come from explicit user ratings after watching a movie or listening to a song, from implicit search engine queries and purchase histories, or from other knowledge about the users/items themselves. Sites like Spotify, YouTube or Netflix use those data in order to suggest playlists or to make video and movie recommendations, respectively. Recommendation systems providing personalized suggestions greatly increases the likelihood of a customer making a purchase compared to unpersonalized ones.

There are different ways to recommend an item to a specific user, but the most commonly used approaches are Content-Based techniques, Collaborative Filtering techniques and Matrix factorization.

Content-Based recommenders try to compare using the features of items such as movie genre or book's authors to recommend similar new items. They can be distinguished as

1. Memory-Based techniques which use the data you have (likes, votes, clicks,...) to establish correlations between items to recommend an item i to a user u who has never seen it before.

2. Model-Based techniques utilize several machine learning algorithms to train on the vector of items for a specific user, so that the model can predict the user's rating for a new item that has just been added to the system. This technique, basically, extracts some information from the dataset, and uses that to build a model to make recommendations without having to use the complete dataset every time. This approach potentially offers the benefits of both speed and scalability.

Collaborative Filtering, on the other hand, recommends items to the user based on what other similar users liked (User-User recommendation), or based on similar items (Item-Item recommendation).

Finally, the Matrix Factorization technique looks at the data from both the users and the items perspective at the same time. A particularly useful feature from this technique, is that the outcome matrix (containing all ratings for all combinations of users and movies, for example) is split into two smaller matrices, reducing the amount of parameters. This makes working with large datamatrices a bit more feasible.

Problem Statement

In this report we worked with the *MovieLens* 10M dataset that can be fetched from the website *http://grouplens.org/datasets/movielens/*. This set consists of collections of Users, Movies and Ratings; it contains about 10677 movies, rated by about 69878 users. Users can rate a movie from 1 to 5. We applied multiple recommender system-approaches to this data and the purpose of this report is to describe the experiments we performed.

Results will be given in terms of accuracies achieved and performances such as the actual run time and memory usage. The report is organised as follows: initially we will present four naive approaches and concisely discuss about results, after which we will consider a more demanding method, called Matrix Factorization, based on Gradient Descent Algorithm, and compare the results.

In order to avoid overfitting and get a good look at the accuracies of the models on 'unseen' data, i.e. data on which the model was not trained, in all these implementations, we applied the 5-folds cross-validation scheme. Basically, with this structure we randomly split our available data into 5 parts of more or less equal sizes and developed 5 models for each combination of 4 out of 5 parts. Then, each model was applied to the part that was not used in the training process. In other words, the algorithm was used 5 times against random subsets of the same dataset. In this way we were able to generate 5 different estimates of the accuracy; their average is considered to be a good estimate of the error.

The accuracies we provide as results, are given in the form of the **Root Mean Squared Error (RMSE)** and the **Mean Absolute Error (MAE)**. The equations for calculating these are given in equations 1 and 2, respectively. Given accuracies in this report are the mean of the crossvalidation results, as mentioned before, if not specified otherwise.

$$RMSE = \sqrt{\frac{\sum(\hat{y} - y)^2}{n}} \tag{1}$$

$$MAE = \frac{\sum|\hat{y} - y|}{n} \tag{2}$$

Naive Approaches

First, we look at four naive recommender approaches. These approaches are fairly simple to implement and work quite well. The goal of these approaches is to estimate the rating a user would give to a specific movie, so that a system can give a recommendation suited for the user. The four approaches that are tested here are approaches using average ratings. In other words, all approaches use the given data, finding averages of either all ratings or specific ratings and using those as estimates of the ratings. Below are the four general equations used for the four naive approaches:

$$R_{global}(User, Item) = mean(All\_Ratings) \tag{3}$$

$$R_{item}(User, Item) = mean(All\_ratings\_for\_item) \tag{4}$$

$$R_{user}(User, Item) = mean(All\_ratings\_for\_user) \tag{5}$$

$$R_{user}(User, Item) = \alpha * R_{user}(User, Item) + \beta * R_{item}(User, Item) * \gamma \tag{6}$$

Global average rating

For the first, and simplest, approach, the global average rating is used as the estimate for all ratings. The result is not very reasonable, since it is very unlikely that each user gives exactly the same rating for each movie. However, it is still more likely to perform better than when using a number that is not chosen based on the given data at all.

We implemented the algorithm in this way: after loading the dataset we created and randomly shuffled a list of values from 0 to 4 as long as the number of ratings. Then for each fold we separate the data between training and test data and calculate the mean of all ratings which will be used for the RMSE and MAE. The final result is an average of the RMSEs of each fold.

Average rating per Movie

The second approach uses the average rating for each movie as the estimate for a rating. While this means that it would take more time to run, since it has to calculate each average rating for each movie, the result should, intuitively, prove more accurate. If a movie is generally considered good, for example, it has won a lot of prizes, it is quite likely to have an averagely higher rating than the average global rating. These differences are captured in this approach and thus likely performs better than when only using the global average mean.

Due to the crossvalidation, it is possible that a training dataset does not contain ratings for a specific movie, since all ratings for that movie are in the testset. In this case, we used the global average as estimate. For the next two approaches, this was done as well, if necessary.

Here the initial part of the algorithm is similar to the global average one. After separating the data for each fold between training and test set, we calcultate the mean of each unique movie. We define the overall mean of all users/movies which will be used in case there are no ratings for a specific movie. Finally we calculate the RMSE and MAE using the mean of each movie.

Approach	RMSE on trainset	RMSE on testset	MAE on trainset	MAE on test-set	Time elapsed (seconds)
Global average	1.060	1.060	0.856	0.856	20.7
Movie average	0.942	0.944	0.737	0.738	69.3
User average	0.970	0.978	0.762	0.769	69.7
Linear regression	0.871	0.879	0.675	0.681	119.0

Table 1: Mean RMSE and MAE scores for four naive recommender system approaches, rounded to three decimals, and their elapsed times for one run in seconds.

Average rating per User

The third approach is very similar to the second approach. This approach, however, uses the mean ratings for each user instead of each movie. If the approach is similar, then the results are likely similar as well. There is a difference, however, which is that there are more users than movies. This denotes that there are more different means, but those means are based on less ratings.

The implementation of this approach is referred to the previous one, since the only difference is that here we use the mean for each user instead of the mean for each movie.

Linear combination of the 2 averages

The last of the approaches is a combination of the two. Using a linear combination of the average ratings of the users and the movies, we can use linear regression to find the regression coefficients to approach the ratings with as low as possible error. The linear equation is then as shown in equation 6.

After finding the coefficients for which this model fits best, so for the coefficients giving the closest approximate rating over all ratings, this approach is likely the most accurate of the four approaches. It takes in account the differences in average ratings for users and for movies and combining the relevance of the two in a single model. If the average ratings for users are more predictive for the actual rating, then the coefficient for this can be relatively high compared to the coefficient for the average ratings per movie.

Since this approach is a combination of the previous two, its implementation includes both users' and movies' means. However, to calculate the coefficients α, β and γ we create a matrix X with three columns: [*Usermean*, *Moviemean*, v], where v is a vector of ones. We transpose this matrix and calculate the coefficients that best approximate the vector of ratings. The predictive ratings are then evaluated as the sum of the product of the X matrix and the coefficients. Finally we determine the RMSE and MAE using the predicted values just computed.

Results and discussion

The average errors, measured in RMSE and MAE, of each approach is shown in table 1 . These are the averages over the five folds of the crossvalidation. The numbers are rounded to three decimals for readability.

The best performing approach here is the linear regression approach. Its error scores are the lowest of the four naive approaches. This is as expected, since it combines the advantages of the two approaches using means for users and movies into one approach, naturally giving an improved result.

The least accurate is the first method, only using the global average rating. While this is even more naive compared to the other approaches, it still is an improvement when compared to using, for example, the number three (the average of the possible ratings one to five) as estimate. When using three as estimate, the RMSE was 1.178 and the MAE was 0.944, so using the global average as an estimate is still not too bad. Using three here is just an example to compare the errors if we do not take in account the previous data, but just use the average of one to five. In this way, the effectiveness of this approach can be slightly more highlighted.

The final two approaches are in between, with the approach using averages for each movie scoring a bit better. This is possibly due to the lower number of movies, making the means a more accurate estimate for each movie, while there are users with very few ratings given, making this mean a less reliable estimate. Also, there could be less variation in how a movie is rated, compared to how a user rates movies. A movie generally seen as bad probably has mostly low ratings, but a user might not be very inclined to give mostly low ratings. This way, the average rating for users might not be as informative as the average rating for movies.

As for the time and memory needed for each approach, the global average approach is the fastest and requires the least amount of memory. Apart from a few variables storing a negligible amount of values, there are only a few variables storing an amount of values linearly related to the number of user-movie-rating combinations, in other words $O(n)$.

The time needed is also described with $O(n)$, since the number of operations needed also linearly depends on the amount of ratings. It only involves subtractions, squares, means et cetera for n values.

The second and third approaches are similar, they both reiterate over the data to find the means for users or movies, resulting in a time needed of $O(n^2)$. The memory needed is still proportional to $O(n)$, since there are no variables with an increase in memory size that is not linearly related to the amount of ratings.

The last approach, using linear regression, is a combination of the previous approaches. Since it does not have an extra reiteration of the data within a reiteration, it is still $O(n^2)$ for runtime (it needs more time than the previous approaches, but it is still proportional to a quadratic increase). The memory needed is also still linearly proportional, so $O(n)$.

Matrix Factorization

For the Matrix Factorization approach we implemented an algorithm based on a paper by *Takács et al.* [1]. The rating datamatrix is denoted by $\boldsymbol{X} \in \{1, ..., 5\}^{I \times J}$, where an element x_{ij} stores the rating of the j-th movie provided by i-th user. I and J denote the total number of users and movies, respectively.

In Matrix Factorization, we want to approximate the matrix $\boldsymbol{X}$ as the product of two matrices:

$$X \approx \boldsymbol{UM} \tag{7}$$

where $\boldsymbol{U}$ is an $I \times K$ and $\boldsymbol{M}$ is a $K \times J$ matrix. The u_{ik} and m_{kj} values represent respectively the k-th feature of the i-th user and the j-th movie.

Let's denote $\hat{x}_{ij}$ as our estimate of how the i-th user would rate the j-th movie. It is given as as the sum of product of u_{ik} and m_{kj} for each k. In other words, we try to estimate the rating a user i would give a movie J, based on the product of k features of a user and k features of a movie. These features can indicate anything, but to get an idea, here is an example. If the first feature for the movies scores high for movies with a lot of action in it and a user generally ranks movies with action in it relatively high, this user will probably tend to like other action movies as well. In this case the first feature of the user would indicate whether the user likes action movies (high value), or dislikes them (low value). If we combine multiple of such features for genres, we can probably predict the rating the user would give a movie quite well, based on which genres he/she likes. However, a feature can still indicate anything, whether the interpretation of it is intuitive or not.

When training this model on the data, we need to establish a way to improve it, in other words, to update the two matrices $\boldsymbol{U}$ and $\boldsymbol{M}$. This is done by calculating the errors e_{ij}, which are calculated by taking the difference between the actual rating x_{ij} and the predicted rating $\hat{x}_{ij}$. The ideal model would have errors as low as possible, so the goal is to minimize the squared errors (SEs).

Gradient Descent with Regularization

In order to minimize the SE (which is equivalent to minimizing the RMSE) we applied the gradient descent method, also as according to the earlier mentioned paper [1]. With this approach we update the weights in $\boldsymbol{U}$ and $\boldsymbol{M}$ to decrease the error, thus better approximating x_{ij}. With a regularization factor λ we prevent overfitting. The specific equations for updating the values in $\boldsymbol{U}$ and $\boldsymbol{M}$ are shown below.

$$u_{ik}^{'} = u_{ik} + \eta \cdot (2e_{ij} \cdot m_{kj} - \lambda \cdot u_{ik}) \tag{8}$$

$$m_{kj}^{'} = m_{kj} + \eta \cdot (2e_{ij} \cdot u_{ik} - \lambda \cdot m_{kj}) \tag{9}$$

where η is the learning rate. In our algorithm, we initialize the value of the parameters as follows:

$$\lambda = 0.05, \eta = 0.005, iteration\_criterion = 0.001, number\_iterations = 75$$

General implementation

We implemented the total algorithm in the form of some nested loops. While these tend to be fairly slow, there are not too much feasible options. You could work with large sparse matrices, but these can use up a large amount of memory.

After setting the seed, initializing some variables and determining a few parameters, the first loop is as used in all implementations for this report, a loop for the cross validation. Within this loop, we need to run the gradient descent algorithm, after initializing all values and performing relevant actions such as splitting the data in training and test sets. The gradient descent is implemented as described in equations 8 and 9, but since there are multiple movies j for each user i (equation 8) and vice versa (equation 9), we take the mean of all the products, since we want to include the information on all movies/users. So, if we want to update the value $u_{1,1}$, we used $\sum e_{1,j} \cdot m_{1,j}/\#movies$, with the sum indicating the sum over all movies for that user, where equation 8 would indicate $e_{1,j} \cdot m_{1,j}$.

This way, we can calculate the RMSE and MAE for all actual ratings x_{ij} and predicted ratings $\hat{x}_{ij}$ in all folds. Taking the average of that over the folds gives us the final results.

Results

Looking at the resulting errors (table 2), the model is actually in the middle between the second naive approach and the third one, the approaches based on equation 4 and 5, respectively. With matrix factorization being a far more sophisticated approach, the resulting accuracy is fairly low. There are multiple possible reasons for this, such as maximum number of iterations being too low. If the algorithm has not really converged yet, the resulting error could still be lower. The same could be said for the learning rate, if this parameter is increased, the algorithm takes bigger 'steps' to the minimum error, reaching it faster.

However, if the algorithm has actually converged, then it is possible that the descent ran into a local minimum. Such a minimum has a higher error value than the global minumum, meaning that there is still room for improvement.

While we try to avoid the second problem using random initialization of the $\boldsymbol{U}$ and $\boldsymbol{M}$ matrices and using cross-validation to test the process five times, it is still possible that there are a lot of unfavourable local minima, making it very unlikely to find the global minimum (or a local minimum approaching it).

Another possibility is that we need more features k to be able to predict ratings better. A higher value for k lets the model identify more features relevant to the prediction process, likely increasing the prediction accuracy.

We also (briefly) tested these possibilities. The results are shown in table 3, with the first column indicating which parameter was changed into which value. The other parameters for each test remained the same as the initialized values mentioned under 'Gradient Descent with Regularization'.

For the four tested changes in parameters, actually none show a better result, compared to the initial parameter values. The first test was for a higher learning rate. While the mean errors do not show a better result, the errors for each fold differ quite a bit. This shows that the result depends more on randomness, probably indicating different local minima. The lowest error here (e.g. the RMSE on traindata was 0.941) is still higher than the error found for the initial run (as shown in table 2).

Next, we discuss the features. We tested both for a smaller and bigger number of features (half and twice the initial amount, respectively). Both results are very similar to each other, both performing worse than the initial value of $K = 10$. The main difference is that the time needed for one run is quite a bit lower for $K = 20$. The amount of iterations needed for the gradient descent algorithm was found to be lower, decreasing the time needed. It was not very clear why this is the case.

The last changed parameter, a lower regularization parameter lambda, did not cause too much of a difference. While it performs slightly worse compared to the initial results, it is not by a lot. Maybe a larger change in results could be accomplished by decreasing the lambda parameter by a larger amount, but this might not improve the model. It could let the gradient descent converge faster (in equations 8 and 9 a lower lambda means a larger update each step), but that does not necessarily mean that the converged value is a lower error.

For recommender systems the memory estimations and runtime play an important role in terms of performance and efficiency. For the Matrix Factorization these estimations depend on the elements of the matrix *X*, in this case Ratings R, Users U and Movies M. In order to minimize the error we used the gradient algorithm, which update every single value u_{ik} in U and m_{kj} in M. Approximately, the time-memory requirement in gradient descent is about O(R*N*M*F) where F is the number of folds. Therefore,

Approach	RMSE on trainset	RMSE on testset	MAE on trainset	MAE on test-set	Time elapsed (minutes)
Matrix Factorization	0.938	0.948	0.725	0.732	24.3

Table 2: Mean RMSE and MAE scores for matrix factorization, rounded to three decimals, and the elapsed time for one run in minutes.

Changed parameter	RMSE on trainset	RMSE on testset	MAE on trainset	MAE on test-set	Time elapsed (minutes)
$\eta = 0.01$	1.036	1.050	0.803	0.814	28.9
$K = 20$	1.008	1.017	0.806	0.812	18.0
$K = 5$	1.006	1.015	0.778	0.785	30.8
$\lambda = 0.01$	0.943	0.953	0.729	0.737	23.4

Table 3: Mean RMSE and MAE scores for matrix factorization after changing one of the parameter values, rounded to three decimals, and their elapsed times for one run in minutes.

we would expect that the required cpu-time and memory would grow linearly with the number of movies, M, the number of users, U, and the number of ratings, R.

Conclusions

This report introduced a few methods to implement recommender systems, specifically for a Netflix movie-user-rating dataset. The four naive approaches show decent results. The approaches using the global average, user-based averages, movie-based averages and linear regression combining previous approaches show increasingly better perfomance in that order. The time needed also increases in that order, but for the second and third approach they are very similar. The linear regression method thus performs the best, but also takes the most time, while the worst performing method is the one using the global average. However, it is by far the fastest method.

The results for matrix factorization indicate that it does not perform as well as the naive linear regression method and it takes a lot more time to run. A few tested changes to the parameters did not improve the model, but the results from these changes are not enough to say much about the effects of changing the parameters on the model.

References

[1] Gábor Takács, István Pilászy, Bottyán Németh and Domonkos Tikk. *On the Gravity Recommendation System.* Section 3.1.