Advances in Data Mining 2018: Assignment 1

September 25, 2018

1 Introduction

The goal of this assignment is to create and experiment with recommender systems for movie ratings. Streaming services like Netflix offer movies and series to their customers. Such companies use algorithms to recommend movies and series to the people who are most likely to watch the offered products, in order to maximize profit. These algorithms are so called recommender systems, which give a prediction of the ratings that users would give to certain movies. In this assignment we implemented five of those algorithms in Python. Subsequently, the accuracy is estimated using the Root Mean Squared Error (RMSE) and the Mean Absolute Error (MAE). The objective was to provide the algorithms for four naive approaches as well as the algorithm for a more advanced approach, which will all be explained in the Methods section. In order to get reliable results, the experiments were done using 5-fold cross-validation. For the complete assignment as it was provided, see [1].

2 Methods

2.1 Dataset

In order to create and test the recommender systems, we made use of the *MovieLens 1M* dataset fetched from `http://grouplens.org/datasets/movielens/`. The dataset contains about one million ratings given to about 4000 movies by approximately 6000 users. In addition to movie ratings per user, the dataset contains some basic information about the users (id, gender, age, occupation) and movies (id, genre, title and production year). However, for this assignment we only used movie id, user id and the given ratings.

2.2 Cross-validation

During our experiments we took the reliability of our results into account by running the recommender systems according to the k-fold cross validation technique, as part of the assignment. Cross validation is a technique that can be used to increase the reliability of results by splitting the dataset in multiple parts at random, using one part of the dataset as training set on which a model can be build while using the remaining data as test set to estimate the accuracy of the model. We split the data into five folds by creating an array filled with randomly assigned indexes for 1/5th of the ratings in the dataset. The ratings belonging to the corresponding indexes were placed in the test set, whereas the remaining data was assigned to the training set. By doing so five times, we created five models with randomly assigned training and test sets over which the average estimated accuracy could be calculated. In our experiments, we used 17 as random seed to shuffle the indexes. We used the same random seed for all of the approaches.

2.3 Estimating Accuracy

To determine the quality of the different recommendation algorithms, we used two measures: Root mean square error (RMSE) and Mean absolute error (MAE). These measures give an indication of performance quality by addressing the difference between the known ratings and the values as predicted by the algorithms. The Root mean squared error measure formula is as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} \left(\hat{y}_i - y_i\right)^2}{n}}$$

First, the difference between the value for rating as predicted by the algorithm (ŷ) and the actual value (y) is calculated. The corresponding value is transformed by taking the power of 2 in order to have positive values to work with. Subsequently, the differences for all the ratings are summed and divided by the total number of ratings to give an estimate of the quality of the algorithms performance.

The Mean absolute error measure differs from the RMSE measure as it does not use the square of the differences in order to get positive values. The formula for calculating the mean absolute error is as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

Instead of squaring the rating/prediction differences, the MAE simply computes the absolute differences. The absolute differences are then summed and multiplied by 1/n, similarly to what happens in the RMSE formula.
As both measures are a reflection of the prediction error, lower values for the measures correspond to more accurate predictions. Therefore, algorithms with lower estimated accuracy values are more likely to be accurate prediction algorithms.

2.4 Algorithms

The implemented algorithms should provide a prediction of the ratings for all user/movie combinations. In total, five algorithms were implemented using Python. The first four algorithms that are shown here are the so called 'naive approaches', which are estimating techniques that use data that is already known from examples in the dataset in order to predict unknown values without adjusting them or attempting to address possible causal factors. In this section we explain the five approaches and show how we implemented them in Python. As we only needed a selection of the dataset, we first made a numPy array 'ratings' in which we stored only user id's, movie id's and the corresponding ratings. This array was used in all of the algorithms to do our calculations.

2.4.1 Global average

The first naive approach predicts the user ratings for movies based on just the global average of all given training ratings. This means that the ratings for every possible user/movie combination are predicted to have the same value, which is the global average of all known ratings. In order to implement this algorithm, we received a **demo script** which already contained the implemented approach The average rating over all combinations was calculated using the numPy mean function. The accuracy was then estimated by simply calculating the difference between all of the known ratings and the global average rating.

2.4.2 Average rating per movie

For this approach, the prediction for the ratings was done by calculating the average rating for each movie, and compare the actual ratings to these values. This approach should be more accurate than the global average approach, as more information is being used. Most likely movies will be rated similarly by the users as good movies are expected to get higher ratings whereas bad movies are expected to be rated lower by all users.
In order to be able to calculate the average rating per movie, we first created a new NumPy array containing the same data as the 'ratings' array, and sorted it on movie id using the numPy argsort function. Subsequently, we created a new array containing the indexes of the ratings in this array where the movie id changed, using the NumPy bincount function. As soon as we knew these indexes, we used the indexes to distribute ratings over distinct movie arrays, so that every array contained only the ratings belonging to a single movie. Whenever a movie was not present in the training set, the value 0 was added to the array. For all of the distinct arrays, we used the NumPy cumsum function to calculate the cumulative sum, after which we divided it by the number of ratings in the arrays to get an average rating per movie. If there were no ratings for a given movie, the global average rating was given as predicted value.

2.4.3 Average rating per user

This approach is very similar to the average rating per movie approach, however, instead of calculating the average rating per movie, the average rating per user was calculated. Therefore, a prediction of the ratings was made based on the previous ratings given by users. This approach is build on the assumption that users are likely to give similar ratings to different movies, such that the users give higher or lower ratings to movie in general. The implementation of this algorithm was the same as for the average rating per movie approach. The only difference is that we first sorted the 'ratings' array on user id instead of movie id, and used the distinct user id indexes to create user arrays, instead of using movie id's.

2.4.4 Linear combination of averages

The final naive approach predicts ratings based on a combination of the three before mentioned averages using linear regression. We used the three average values as input attributes in a linear regression model to find the best combination for predicting the ratings. To do so, we created an array with average ratings per movie and an array with average ratings per user. Again, if a movie or user average was missing, it was given the global average rating. For the global average no array had to be created, because it is a constant factor. The linear regression model was build using the NumPy linear algebra least squares function. The user average as well as the movie average were given as input for the model as variable regression parameters, whereas the contribution of the global average was modelled as single parameter since it does not depend on specific user/movie combinations.

2.4.5 Matrix factorization with gradient descent

Define the ranking matrix X comprised of elements x_{ij} representing the rating with rows i representing the user-ID i and and the columns representing the movie-ID j belonging to that rating. Not every user has rated all movies, so the ranking matrix is not filled. Since there are roughly 6000 users and 4000 movies the total number of elements of X is around 24 million and only 1 million of elements are actually given/known. The goal of the Matrix Factorization method is to provide a prediction for each user-ID and movie-ID combination. The way this is attempted is by assuming that the matrix X can be approximated by the product of two matrices U and M:

$$X \approx \hat{X} = UM$$

The indices of U and M are i x k and k x j respectively, where i and j again are the user-ID and movie-ID respectively and k is the number of different features that the user and movie are expressed in. The idea behind this factorization is that each user and movie can be expressed by k different features that each represent some aspect of the movie and user. The bigger the feature is an aspect of the user or movie the higher the value for that feature. The total state of the user or movie can then be written as a vector of length k where the elements represent the prominence of different features:

$$\boldsymbol{u_i} = (u_{i1}, ..., u_{ik}),$$
$$\boldsymbol{m_j}^T = (m_{1j}, ..., m_{kj}).$$

The inner product of the vector for the user and movie is then equal to the prediction rating the user would give the movie:

$$\hat{x}_{ij} = \sum_{f=1}^{k} u_{if} m_{fj},$$

the more similar the user and the movie, the higher the predicted rating $\hat{x}_{ij}$ will be. The number of features k are set manually. What these different features actually are is not determined or clear upfront. The features will be based on the data by training the prediction matrix so that it gives more accurate predictions. We will define some useful quantities needed for this. The error function e_{ij} equal to:

$$e_{ij} = x_{ij} - \hat{x}_{ij} = x_{ij} - \sum_{f=1}^{k} u_{if} m_{fj}$$

is the difference between the actual rating and the predicted rating for a specific (i, j) user-movie-ID combination. We want to minimize the error function hence a more useful quantity would be:

$$e_{ij}^2 = (x_{ij} - \sum_{f=1}^{k} u_{if} m_{fj})^2,$$

since it is always positive and we can use gradient descent to find the minimum of this function. The training is done as follows:
Step 1: Initialize all values of the User and Movie matrices U and M at random by a value between 0 and 1.
Step 2: Iterate over each rating in the training data and compute what the product of the matrices gives for the user-ID movie-ID combination belonging to that rating. Now of course in the beginning this will most likely be completely off the actual rating, however as we iterate over the ratings and adjust the values of the matrices towards a direction that minimizes the squared error function the predictions will become better. Using the gradient of the squared error in combination with a regularization term λ to prevent overfitting we multiply the value with an appropriate learning rate η to find the adjustment value and subtract it from the matrix values to guide it towards the minimum:

$$u_{ik} = u_{ik} - \eta(\frac{\partial}{\partial u_{ik}} e_{ij}^2 + \lambda u_{ik}) = u_{ik} + \eta(2e_{ij} u_{ik} - \lambda u_{ik}),$$
$$m_{kj} = m_{kj} - \eta(\frac{\partial}{\partial m_{kj}} e_{ij}^2 + \lambda m_{kj}) = m_{kj} + \eta(2e_{ij} m_{kj} - \lambda m_{kj}).$$

Step 3: keep iterating over the whole training data until satisfied with the accuracy of the predictions.

2.5 Hardware

We experimented with the algorithms on a computer with an Intel i5 processor and 8 Gigabytes of working memory running on a Linux operating system.

3 Results and interpretation

After running the experiments we got the following results. In order to interpret and compare the results of the algorithms, we report the two average estimated accuracy measures for both the training and test sets, actual run time and required computational time and memory for all algorithms separately. We express the required amount of time and memory with help of the big O notation, which is a standardized form to express how quickly the time and memory would grow as the input grows. To be able to compare the required time and memory of the different algorithms with each other, we did not take the cross-validation and accuracy estimation procedures into account when interpreting the big O notation. Also, we assume the worst case scenario in terms of required time and memory. The parameters M (movie), U(User) and R(Ratings) are used to describe the time and memory. To check whether we conducted our experiments properly, we compared our results with the results reported on http://mymedialite.net/examples/datasets.html, since the experiments have been done before by other people.

3.1 Global average rating

The measured actual run time for the global average rating algorithm was 2.92 seconds. The estimated accuracy measures were as follows:

RMSE	Training set	1.117
	Test set	1.117
MAE	Training set	.934
	Test set	.934

Table 1: Estimated accuracy measures

To calculate the global mean we only need to scan the data once, summing up all the ratings (one float, Sum) and counting them (one integer). The global mean is then S/N, so we need memory to store only 2 numbers, i.e., the required memory is O(1). The required time would be O(R) since the required time grows linearly as the amount of ratings increases.

3.2 Average rating per item

The measured actual run time for the global average rating algorithm was 5.58 seconds. The estimated accuracy measures were as follows:

RMSE	Training set	.974
	Test set	.979
MAE	Training set	.778
	Test set	.782

Table 2: Estimated accuracy measures

To calculate the average rating per item, we assume the worst case scenario in terms of total amount of memory and time needed, which would be that every movie is rated at least a single time. This would result in one average value for all of the movies. For this we would need the summed ratings (one float, Sum per movie) and the count of ratings per movie (one integer per movie). Therefore, we need to store 2 numbers per movie, which makes the required amount of memory grow linearly in regard to the amount of movies, so the required memory is O(M). The required time of this operation is O(U*M), because for each item/movie it needs to go over all the users to see which rating they gave the movie in order to get the average.

3.3 Average rating per user

The measured actual run time for the global average rating algorithm was 5.40 seconds. The estimated accuracy measures were as follows:

RMSE	Training set	1.028
	Test set	1.035
MAE	Training set	.823
	Test set	.829

Table 3: Estimated accuracy measures

The required memory and time can be derived in the same way as with the average rating per item. However, instead of using M as input, time and memory can be expressed using U. The required memory therefore is O(U). The required time of this operation is O(U*M), because for each user it needs to go over all the items/movies to see which ratings the user gave to them in order to get the average of the user.

3.4 Linear combination of averages

The measured actual run time for the global average rating algorithm was 9.27 seconds. The estimated accuracy measures were as follows:

RMSE	Training set	.915
	Test set	.925
MAE	Training set	.725
	Test set	.733

Table 4: Estimated accuracy measures

As the global average rating is provided to the model as constant parameter, the required time and memory mostly dependent on the user id's and movie id's and the corresponding ratings. With use of the user and movie average rating arrays along with the predicted ratings, the best matrix is found in order to minimize the squared error between the prediction of the model and the real proportion between the parameters by creating a coefficient matrix. Therefore, the required memory is O(M+U+R). The required time of this operation is O(R) since most of the time is spent on the linear regression problem and there are Rx3 elements in the matrix that needs to be fitted.

3.5 Matrix factorization

Using the values for the regularization $\lambda = 0.05$, learning rate $\eta = 0.005$, total number of iterations 75 and number of features $k = 10$ the following results were obtained. The measured actual run time for the global average rating algorithm was 1 hour, 16 minutes and 3,15 seconds (4563.15 seconds).

The estimated accuracy measures were as follows:

RMSE	Training set	.770
	Test set	.873
MAE	Training set	.604
	Test set	.681

Table 5: Estimated accuracy measures

The algorithm iterates only over all the ratings when its computing how to update the elements in the user and movie matrix, so having 10 times more ratings would require 10 times more time regardless of how many movies or users there are. So the required time to run this algorithm is O(R). The memory of the algorithm does depend on the number of movies and users since the memory is needed for saving the numbers in the User matrix which has dimensions (number of users x number of features) and Movie matrix which has dimensions (number of features x number of movies). So the algorithm requires O(M+U) memory.

4 Conclusion

From the results of the previous section, the following order of accuracy for the different method applies: Matrix Factorization > Linear Regression > average movie rating > average user rating > global mean rating.

The global mean rating (gmr) being the least precise was to be expected, because the only information that is in the value of the gmr is how movies are rated on average. It does not take into account any reasons that for a specific movie and user this rating can be off. For example the movie might be a niche and is not well approximated by the average or the user might just be a person that rates everything high.

A better approximation would be to look at the history of the user and use that as the prediction: if for example a user tends to rate a lot of movies with a certain value then it might be likely that he rates a new movie in a similar fashion. However this might not work well at times: for example lets say a user has always rated movies of a particular genre and now he is rating a movie of a completely different genre. Then the rating of a certain genre of movie might not be reflective and a good prediction of a different genre of movie, because the user might simply not like it as much. Another reason might be that the user has always watched really good movies and now watches a bad one.

Looking at the history of movie ratings actually has a very similar set of advantages and disadvantages as the history of user ratings. However the reason it works a little better might be because the pool of different types of users that rated the movie is deeper than the different types of movies the user has rated thus being a better predictive value.

Using linear regression to find the ideal linear combination of these values of course will be better than the individual predictors, because the least squared error value is found in how to combine them optimally and each of the predictive qualities are combined in an ideal way.

Finally the matrix factorization method is similar to the linear regression method in the way that it takes into account both user and movie information. However the matrix factorization doesn't just look at the average ratings of a user or movie, but at each individual rating that is given and adjusts its prediction slightly on each one of them. This explains why it does better than all the other methods, but the downside of looking at each individual rating is that it takes much longer to run the method.

The total number of possible ratings is roughly equal to 6000x4000=24 million and the number of known ratings is around 1 million. One of our concerns before applying the matrix factorization method was that perhaps there were too few ratings known compared to the whole set of possible ratings thus making the method not as accurate. This was not the case however as the matrix factorization method was the most accurate method. Apparently the method is so powerful that even only knowing 1/24 of the possibilities already gives you enough information to get an accurate factorization and thus an accurate predictor.

References

[1] Kowalczyk, W. J. (2018). *Assignment 1: Recommender Systems.* Retrieved September 12, 2018, from https://blackboard.leidenuniv.nl