Delft University of Technology

Data Mining
CSE2525

Netflix Challenge Report

Authors:
Luca Becheanu 4996240
Alexandru Lungu 5069602

Kaggle IDs: lucabecheanu, alexandrulungu

January 22, 2021

1. Introduction

The goal of this challenge was to implement a movie recommendation system on a dataset that consists of 910,190 ratings (on a 1 to 5 scale) given by 6,040 users to 3,706 movies. Next to the ratings, the data contains some additional information on the users (gender, age, and profession) and on the movies (title and year of release). Based on this data, our algorithm has to predict as good as possible what rating a particular user will give to a particular movie.

The problem we had to solve involved making predictions of the user-movie ratings provided. In order to tackle this, we kept the original size of the data and we used simple methods of computation. For each algorithm that was used to make the predictions, we measured the root mean squared error (RMSE) of the predictions and created tables to illustrate the efficiency and accuracy of each of those. The data we used consisted mainly of the provided ratings, but for some algorithms we had to predict the missing ratings as well in order to create better estimates for our final predictions.

To summarize our main contributions to solving this problem, we developed algorithms to recommend movies by making predictions on provided ratings and measuring their accuracy and time efficiency.

2. Methodology

Step 1

2.1 Collaborative Filtering User-User (CFUU)

This algorithm works by finding the most similar users using a nearest neighbor approach with the neighborhood size set to 100. Based on the users found, we make a prediction of what the rating of this user for every movie we had to predict would be. To find the most similar users, we had to use a similarity measure. First, we tried using cosine similarity, but found Pearson correlation to be performing better. We started by normalizing the data (computing the mean vector by getting the mean of each row and subtracting each non-zero element of that row by that value) so that the values are centered around 0. Then, we saved the correlation matrix in a file so that it would not be necessary to compute it each time the program ran. Once we have the correlation matrix, we can start finding the nearest neighbors and predicting the ratings. First, the matrix needs to be sorted on each row by the most similar users first. Then, we take the first 100 users that do not have missing ratings for that movie and compute the weighted average of the ratings and similarities. Finally, it is worth mentioning that the predictions could sometimes turn up being less than 1 or more than 5, so we had to bound the predictions in that range. There are also cases in which a rating could not be computed (such as when division by 0 would occur) and for those we set the prediction to 3 as it is the median of 1 and 5. We improved this later in CFII by setting that value to the average movie rating, instead of just 3.

2.2 Latent Factors (LF)

The first step was to add the known ratings to a rating matrix, then normalizing the rows (center values around 0, subtracting each row by their mean) to achieve a faster computation time. Moreover, we used the numpy.linalg.svd function to compute the U, Σ and V^T matrices. We took Q=U and P=Σ*V^T. To ensure the running time is fast and also achieve accurate results, we sliced the matrices to only include the first 50 factors. Afterwards, we made use of the Stochastic Gradient Descent algorithm and repeated it 10 times to make sure the function converges as close as possible to 0. The result of the predictions is then the dot product of the row of Q (Q_i) and column of P (P_j) for each movie situated at (i, j) in the predictions file, plus the mean value of row i.

Step 2

2.3 Collaborative Filtering Item-Item (CFII)

Item to Item Collaborative Filtering works almost exactly the same as its User to User counterpart, except now we compare the most similar items instead of the most similar users. For this algorithm, we set the neighborhood size to 15 as it yielded better results than the 100 that we used for the first algorithm. This algorithm ended up being better overall than the User-User variant, as it had a lower RMSE for the predictions.

2.4 Latent factors transpose matrix (LFTM)

This algorithm is exactly the same as LF, the only difference being that we took the ratings matrix to have movies on the rows and users on the columns. This is the same as taking the transpose of the other matrix. As was the case with CFII, this algorithm performed better than its counterpart. It is actually our best algorithm, having a RMSE of 0.87535.

2.5 Baseline Estimate + Collaborative Filtering Item-Item (BECFII)

For this one, we combined the CFII algorithm we used as described above with baseline estimates, more specifically we computed the average of all provided ratings and also took into account user and movie biases. This ended up performing worse than we thought, with a RMSE of 0.96431. It is in fact our second worst performing algorithm on this list, but it also has the most potential to be improved as it should score much higher.

2.6 Latent Factors Transpose Matrix + Global Biases (LFTMGB)

We tried implementing this one, but it ended up performing the worst, having a RMSE of 1.27447. Therefore, we ended up focusing more on the other algorithms.

3. Results

Our first implementation was of CFUU with 3 nearest neighbours and it gave us a RMSE that was way too high. It was also very complex (O(n^3)). Therefore we improved the algorithm, lowering the number of iterations and saving the pre-computed similarity matrix in a file giving us a huge speedup. This brought the complexity down to O(n^2). This time we tried setting the number of neighbours higher since theoretically speaking, the more neighbours, the better the prediction is. However, more neighbors will result in getting more users with low similarity. The result will then be worse since the difference from the target user becomes higher. After multiple attempts, the best score was achieved by having 100 neighbours (Table 1).

The first latent factors algorithm, implemented in step 1, achieves better performance than CFUU. The best score we got was RMSE=0.89192 as seen in Table 2, when we repeated the SGD algorithm 10 times. Initially, we took the first 50 factors and used lambda=0.5 and learning rate=0.0005. After a bit of manual hyperparameter tuning, we found lambda=0.6 and learning rate=0.0001 to be the best values to use for the Stochastic Gradient Descent.

For the implementation of CFII, after performing hyperparameter tuning (Table 3) the best number of neighbours found was, as mentioned before, 15. The reason for having a smaller k this time around is that there exists a popularity bias where the popular movies tend to get a higher prediction score. The results were better than our CFUU approach due to the fact that users have a more dynamic nature, whereas movies usually do not change that much.

Finally, the best results were achieved using LFTM (RMSE 0.87535, Table 4). No matter what parameters we chose here, the duration of this algorithm was always around 5 minutes. However, better performance was reached when we took 25 latent factors instead of 50 like we did for the LF algorithm in step 1. Since this algorithm was the best and CFII performs better than CFUU, we can conclude that an item to item approach performs better than its user to user counterpart.

4. Discussions, Conclusions and Future Work

In this report, we covered our approaches in developing a recommender system for movies. As much as we would like the implementations to be perfect, there is room left for improvement that can be addressed in future work. We did not manage to properly make use of baseline estimates and global biases since these would have further improved our predictions. However, we managed to obtain a pretty close score to the Netflix Prize requirement and the only thing left for us was to try the “kitchen-sink” approach, combining all our prediction algorithms into one that performs the best.

Tables

No. of neighbours (k)	Approximate time (minutes)	Score (RMSE)
3	15 (first implementation)	1.06624
15	2:30 (improved implementation)	0.95454
100	7	0.94899
200	11:30	0.95455

Table 1: CFUU

No. of iterations	Approximate time (minutes)	Score (RMSE)
10	5	0.89192
20	8	0.89547
50	40	0.92544

Table 2: Latent factors

No. of neighbours (k)	Approximate time (minutes)	Score (RMSE)
13	5	0.88438
15	5	0.88502
100	10	0.91916

Table 3: CFII

No. of factors	Score (RMSE)
25	0.87627
30	0.87535
40	0.87601
50	0.87843

Table 4: LFTM

References

https://www.geeksforgeeks.org/user-based-collaborative-filtering/

https://towardsdatascience.com/introduction-to-latent-matrix-factorization-recommender-systems-8dfc63b94875

http://www.mmds.org/mmds/v2.1/ch09-recsys1.pdf

http://www.mmds.org/mmds/v2.1/ch09-recsys2.pdf