Web Intelligence - Recommender Miniproject

Erik Sidelmann Jensen
ejens11@student.aau.dk

Lasse Vang Gravesen
lgrave11@student.aau.dk

Dennis Jakobsen
djakob11@student.aau.dk

1 Recommender Miniproject

1.1 Data Loading & Manipulation

The structure used to contain the data being loaded is based on Dictionaries, specifically because we are provided with a movie and a user id and it makes sense to structure it like that. Loading is done by first loading the probe data and relevant information is put into a UserRating class that contains everything that is known about that rating(which movie and user and the actual rating). Then the training data is loaded, which is done the same way as the probe data. We only load movies that are represented in the probe dataset, and also restrict it to very few files. Then we manipulate the data such that we can run test the accuracy later, specifically we take the ratings from the training data that are represented in the probe data and add it to the probe representation and then we remove it from the training data. This is done because we want to see how accurately the rating can be estimated, and if the rating is already in the training data when the learning takes place we will not know how accurately it is actually guessing unknown ratings.

The data being manipulated is not resaved because loading data is not a bottleneck.

1.2 Learning

There are two different recommender systems to use, collaborative filtering and content-based filtering. For content-based filtering the content of an item is what you recommend based on. This information is however not present in the Netflix competition data. Only a movie-id, user-id and a rating is present in this data. Collaborative filtering on the other hand is where you predict ratings for users based on other users' ratings, where users that are similar get similar ratings. Because the data contains a large amount of user movie rating pairs collaborative filtering makes a lot more sense than content-based filtering Therefore a collaborative filtering recommender system is built.

The approach we take to CF filtering is called 'matrix factorization'. We do this because it trades more complex offline model building for faster online prediction generation and we are interested in large scale, with missing ratings. Traditional matrix factorization through single value decomposition (SVD), which for large scale is very slow. So we need a variant that approximates SVD, called Funk-SVD, created by Simon Funk for a Netflix recommender competition.

Through Funk-SVD we want to create two matrixes, called A and B that contain only the most valuable information for predicting the ratings. See Figure 1.1 and Figure 1.2.

In an equation what we want from this is:

$$\hat{R}_{mu} \approx \Sigma_{k=1}^{K} AmkBku]$$

We want to find A and B such that we can reproduce the observed ratings as best as we can, given K. We do this by minimizing the squared error through stochastic gradient descent.

The magic is then that from this process, unknown ratings have been filled in aswell.

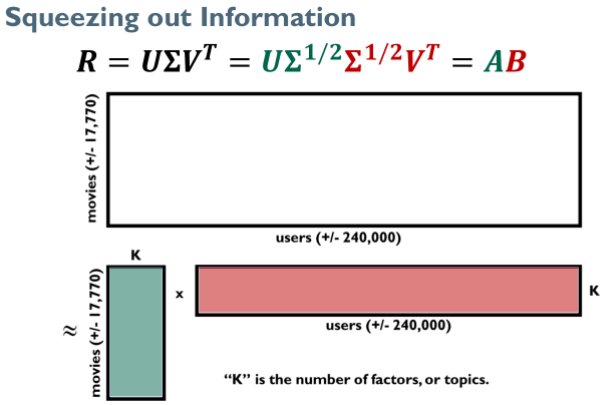

Figure 1.1: Squeezing out information to get A and B

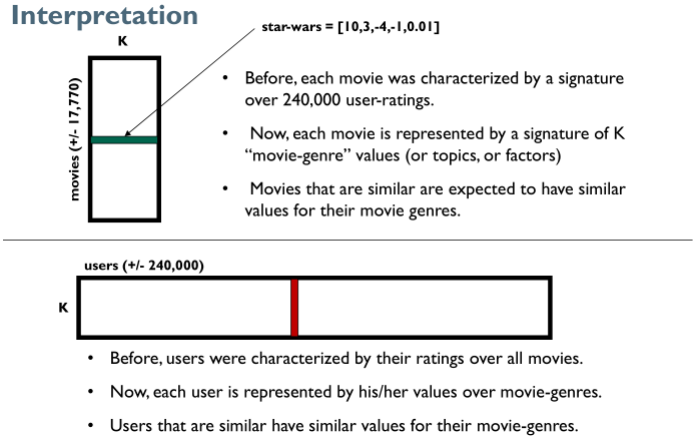

Figure 1.2: Interpretation of A and B

To start, we loaded the data and the next thing we want to do is to remove the movie and user means from the training data, in order to do some pre-processing for the algorithm, see Figure 1.3. We remove these means because we want to normalise the ratings to take into account the differences between peoples enthusiasm when rating. For example, one person could rate a movie with two star, where another person could rate the same movie four stars, but the meaning of the stars would be the same, because the first user has a tendency to rate low whereas the other user has a tendency to rate high.

$$R_{mu} \leftarrow R_{mu} - \frac{1}{U_m}\sum_s R_{ms} - \frac{1}{M_u}\sum_r R_{ru} + \frac{1}{N}\sum_s\sum_r R_{sr}$$

Figure 1.3: Removing the obvious structure from the ratings.

U_m: total number of of observed ratings for movie m.

M_u: total number of observed ratings for user u.

N: total number of observed movie-user pairs.

We construct the movie feature A and the user feature B by first filling it with random values between -0.5 and 0.5. Then we use the stochastic gradient descent algorithm to calculate A and B, see Figure 1.4. This algorithm iterates until the error reaches a local minimum. We do not check the error for every iteration, but instead check it with some interval. It is not necessary to monitor this error for a minimum, since a predetermined amount of iterations can be used, for example 1 000 000, and just adjust this value until a minimum error has been reached.

$$A_{mk} \leftarrow A_{mk} + \eta \sum_u \left(R_{mu} - \sum_i A_{mi} B_{iu}\right) B_{ku} \qquad \forall k$$

"S. Funk": $\eta = 0.001$ good value for Netflix data

$$B_{ku} \leftarrow B_{ku} + \eta \sum_m A_{mk} \left(R_{mu} - \sum_i A_{mi} B_{iu}\right) \qquad \forall k$$

Figure 1.4: Stochastic gradient descent.

k: Latent factors (between 10 and 50).

R_{mu}: Rating for movie m and user u.

η: Learning rate.

At last, we add in the obvious structure from the pre-processing step again, and truncate the calculated rating to fit the Netflix rating scale. Meaning that every rating above five results in five, and all ratings below one results in one.

Alternative algorithms could have been used such as user-based nearest neighbors CF (kNN) and item-based nearest neighbors CF (kNN). Although for these techniques there are issues with scalability when having millions of users and thousands of movies. Furthermore there are a possible problem of coverage where the kNN algorithm does not find enough neighbors and for example with user with preferences for niche products.

Using this model-based algorithm matrix factorisation scales to larger data set. When recommending a movie to a user, only the learned model is used, which makes it much faster, although the model is re-trained or updated periodically.

The model-based approach seems as the best choice for this kind of recommender system, although we did not manage to get useful results out of it.

1.3 Scoring

We evaluated the RMSE(Root-Mean-Square Error) for the probe data and the result ranged between 1.15 and 1.50, where it should roughly have been 0.90.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{r}_i - r_i)^2}{n}}$$

For one subset of the overall dataset we did manage to get good results, specifically using midsized movie files and 25 of them we could get an RMSE score of 0.90, though changing the latent factors seemed not to make much of a difference in the final result.

In all, the results seemed to vary wildly based on what dataset was picked out, and very few of them gave good results. To perfect this, more time would have been required to do different things in the code, but there was none left.