Exploring collaborative filters: Neighborhood-based approach

Minas Gjoka
mgjoka@uci.edu
Student id: 42369077

Fabio Soldo
fsoldo@uci.edu
Student id: 90007558

Abstract

In this project, we study the effectiveness of collaborative filtering mechanisms in the context of the Netflix competition. We focus our attention on a dataset provided by Netflix which includes a training set with more than 100 million 4-tuples: user id, movie id, rating, and date [3]. In the first part of this project, we develop a simple model to predict future ratings of users based on the ratings of the same users on similar movies using k-nearest neighbors (k-NN) methods. In the second part of the project we also implement two of the k-NN methods which were proposed by the currently first team in the contest [1]. We test the performance of the implemented techniques on the whole Netflix dataset.

1 Introduction

In recent years, the importance of automated systems that are able to analyze user interest in specific items and provide accurate future recommendations has grown significantly. Some of the major e-commerce based companies, such as Amazon or Netflix, have made recommender systems a central component of their commercial strategies.

In October 2006 Netflix launched a new competition to further enhance the performance of its recommender system and promote deeper research in the area. As part of the competition, Netflix released a dataset which consists of around 100 millions ratings by users for specific movies. The winner of the competition has to implement an algorithm which outperforms Netflix's proprietary recommender system, Cinematch, in predicting future user ratings by more than 10 percent. As an incentive to encourage participation in the competition the winning team will receive an award of $1.000.000.

The consistent prize and the rich dataset have stimulated much interest in the Netflix competition. Within the first year over 2600 teams posted their algorithm and tried to reach the 10% gap from the Cinematech score. Until March 18th, the best improvement obtained by a team was at 9% (precisely 9.01%). This team, known as BellKor (also previously KorBell), recently highlighted part of its algorithms after winning the progress price. The latter is an annual prize for the team which reaches the highest score at the end of each year that the competition lasts.

The final solution of the BellKor team consists of linear combination of 107 separate sets of predictions. However, retrospectively the members of the BellKor team acknowledged that "it is possible to achieve at least 7.58% improvement with a linear combination of three prediction sets that combine all the key elements discussed in their article: nearest neighbor models, neighborhood aware factorization, and latent factor models". In the project we analyze the performance of nearest neighbor models. We try both to implement some simple but new scheme, and to reproduce the same nearest neighbor model employed by the BellKor team [4].

2 The Netflix Competition

The dataset released by Netflix consist of the ratings created by over 480,000 users on 17,770 movies. Each rating is an integer from 1 to 5. The date when the rating was created, which varies between 1999 and 2005, is also given.

The overall dataset consists of more than 100 million entries. These entries have been partitioned in two parts: a training set, and a hold-out set of about 4.2 million entries. Specifically, the hold-out part is made up of the last 9

ratings provided by each user - or less if a user did not rate at least 18 movies in the temporal period examined. The hold-out part is further (randomly) split in 3 parts, called Probe, Quiz, and Test set. The Probe set is attached with the Training set and can be downloaded together with it. The main purpose of this Probe dataset is to provide participants with an easy way to estimate the performance of their algorithm. The Quiz and the Test set are kept by Netflix and consist of an evaluation set that participants are required to predict ratings for. Once a participant submits predictions, the Netflix system returns the root mean square error (RMSE) on the Quiz set which is defined as:

$$RMSE(dataset) = \sqrt{\sum_{\forall(u,i)\in dataset} (prediction(u,i) - realrating(u,i))^2} \quad (1)$$

RMSE is also posted online in the leaderboard ([6]). However, in order to win the competition, the 10% improvement must be achieved over the Test set, and results over this set are never disclosed by Netflix. This precludes clever systems to somehow figure out which entries are present in the Test set by mean of repeated submissions.

3 Dataset Analysis

In this section we present our analysis of the Netflix dataset. In our analysis we consider the complete Training + Probe dataset, which requires about 2GB of hard disk storage uncompressed. Every entry in the dataset is represented by a 4-tuples: (user_id, movie_id, rating, date). The size of the dataset combined with the characteristics of the data poses an interesting challenge for prediction.

One of the first challenges encountered was the actual manipulation of such a large dataset. For such a huge dataset, tools such as Matlab or even python proved to be too slow in executing even simple algorithms in the amount of time compatible with the constraints imposed by the project deadline. Thus, we had to resort to a low-level C/C++ framework, icefox (available at [5]) to implement basic operations on the Netflix dataset (e.g. fast look-up of user's rating, movie's rating, etc..). Leveraging the API provided by the icefox framework we implemented our algorithms as a set of C/C++ modules which actually extend the icefox framework capabilities. Obviously, making use of the C or C++ language significantly increased the time required to code the algorithms, however, as aforementioned, we tried several programming languages, and this was the only solution which enabled us to perform efficient operations, and data manipulation of such a large dataset.

Another challenge posed by this competition is that, despite the large size of the dataset provided, about 99% of the potential pairs user-movie have no rating! Consequently, many machine learning methods designed to work on dense data are likely to fail in providing good predictions for the Netflix dataset.

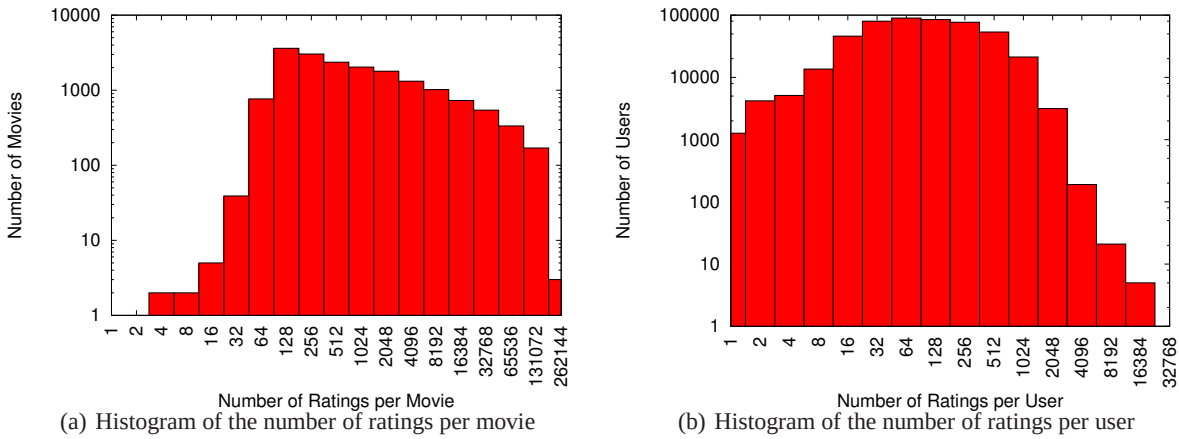
Moreover, both the distribution of ratings per movie, and the distribution of ratings per user is extremely skewed. For instance from fig 1(a), there is a movie with 262144 ratings while the most common cases for movies are those with 128-512 ratings. We also observed that while some movies reached a sort of "general consensus" among the users, having a very small variance, other movies are more polarized, being likely appreciated by a peculiar type of users. These movies, with a large variance, may actually be more informative than movies with a low variance in the process of selecting "similar" users.

Finally, we also observed some unexpected behaviour in the data, such as a sudden increase in the average rating during the Fall Quarter of 2004 [1]. We could not find any satisfactory answer to justify this increase, however, as pointed out in [1] we are not as much interested in explaining why this increase occurred as we are interested in finding a clever way to account for it in our predictions. We anticipated here that, due to the time constraint, and the inner complexity of the challenge we did not face this aspect of the competition within this class project, leaving it for a future study. Here we limit ourself to point out the fact that the temporal dimensional has several characteristic that may be employed to further boost up the performance of the recommender system.

4 Recommender Systems

A recommender system is an automated system which is able to analyze past user ratings and give a prediction on what the future, unobserved ratings would be. More formally, let us denote with U the set of all users and let M be the set of items (movies in our case). Both U and M can potentially be very large, up to millions in certain systems. Let $\phi : U \times M \rightarrow R$ be a utility function which expresses how much a specific user u liked a certain movie m where R is the set of possible ratings. Then, for each user $u \in U$, we want to select the movie $m \in M$ (or more generally, the set of movies/items) such that the user's utility is maximized. More formally,

$$\forall u \in U \qquad m* = \text{argmax}_{m\in M}\phi(u,m)$$



(a) Histogram of the number of ratings per movie

(b) Histogram of the number of ratings per user

Figure 1: Dataset Analysis. For each figure both axes in logscale

In general, each element of the space U can be defined with a profile, that is, a set of labels (e.g. user identifier, age, genre, etc..). The same applies to M. In the simplest case case both the profile of a user and a movie is represented by unique ids.

The central problem of recommender systems is that the utility ϕ is usually defined over a small subset of the total space $U \times M$. The remaining values values needs to be somehow inferred from the available data.

5 Collaborative Filtering

Extrapolations from known to unknown ratings can be done by estimating the utility function that optimizes certain performance criterion, such as, in our case the root square mean error. Once the unknown ratings are estimated, actual recommendations to a user of one or more items are made by selecting the highest ratings among all the estimated ratings for that user.

In general, the new ratings of the not-yet-rated items can be estimated in many different ways using methods from machine learning, approximation theory, and various heuristics. Recommender systems are usually classified into the following three categories, based on the algorithm behind the recommendations :

- Content-based recommendations
- Collaborative recommendations
- Hybrid approaches, which combine both collaborative and content-based methods.

The goal for both content-based and collaborative recommendations is the same: recommend items that people with similar tastes and preferences liked in the past. The difference is that content-based recommendations are made by profiling users or items, through the use of descriptive labels. For instance, in the contest of the Netflix competition a content-based approach will aim to profile all movies based on, for example, genre, main actor, etc.. Collaborative recommendations, instead, aim to find similarities between users and/or movies to predict future user rating.

Since the dataset provided does not include any information beyond the 4-tuples (user_id, movie_id, rating, date), within this project we will focus only on collaborative recommendations.

6 Nearest Neighbor Methods

Probably the most common collaborative filtering method is the family of nearest neighbors. One of its advantages is that while very simple and intuitive at the same it gives insights on the nature of the dataset. The goal of these methods is to find, for instance, an appropriate “neighborhood” of the user and leverage the ratings given by the neighbors of that user to predict unobserved ratings of the user itself. Thus, in order to use these methods we should in general: i) find a representative neighborhood, and ii) find the more convenient set of weights to assign to every neighbor, when predicting unobserved ratings.

More formally, we can estimate the unobserved rating of u on movie i, using a user-based kNN as,

$$r_{ui} = \sum_{v \in N(u)} w_{uv} r_{vi} \tag{2}$$

or, similarly, using an item-based kNN as,

$$r_{ui} = \sum_{j \in N(i)} w_{ij} r_{uj} \tag{3}$$

In this project, we focus more on the latter type of kNN. The theoretical foundations, and the derivation of the best estimates for the former case are analogous. We focus on the item-based version because: i) in [1] the authors showed that in the contest of the Netflix challenge item-based kNN perform better than user-based kNN, and ii) the computational complexity of the user-based is too high to be handled efficiently in the short amount of time allowed to complete the project [1].

6.1 A Simple Algorithm

We first tried to implement our own variant of the kNN algorithm based on our understanding of how users rate movies. To this purpose we compute the neighborhood for every user u and movie i in the dataset. For the similarity between movies we use the cosine similarity function:

$$S_{ij} = 1 - \frac{< \bar{i}, \bar{j} >}{||\bar{i}|| \cdot ||\bar{j}||} \tag{4}$$

where $\bar{i}$, $\bar{j}$ are the vectors which contain the ratings received for movie i and movie j respectively, i.e. if $i = (0, 1, 4, 2)$ it means that it has received rating 1, 4, 2 by users 2,3 and 4 respectively, and no rating from user 1. We note that the complexity of these operation is quite significant in our case: the vectors $\bar{i}$, $\bar{j}$ are in fact defined in an very high dimensional space. We use appropriate data structures to efficiently access these large amounts of data, and exploit the sparseness of vectors to store them efficiently (hash tables). Eq. 4 needs to be evaluated for every pair of movies, that is, over $1.5 \cdot 10^8$ times (due to symmetricity of the function). For each movie i we keep only the 200 closest neighbors (initially decided due to memory constraints). At least 6 hours were needed for such computation in the whole dataset with an Intel Xeon 2.66 Ghz machine. The cosine similarity function eventually outputs a number between 0 and 1 where values closer to 0 are interpreted as more similar.

An analogous similarity function is used to evaluate the interdependencies between different users. For every pair of users, u and v, we use the cosine function:

$$S_{uv} = 1 - \frac{< \bar{u}, \bar{v} >}{||\bar{u}|| \cdot ||\bar{v}||} \tag{5}$$

where $\bar{u}$, $\bar{v}$ are the vector of rating provided by every user (as before, 0 is used to indicate that no rating are given for a specific movie). The computation of the neighborhood for users is much more computationally difficult since we have in total $1.15 \cdot 10^{11}$ unique pairs. Due to such practical constraints, for each user u we randomly sample 15.000 users out of the total 480.000, compute the similarities between all these pairs using Eq 5 and keep only the first 30 most similar users (again due to space/memory constraints).

Given the computed similarities from Eq.4-5 we estimated the unobserved rating of user u for movie i as,

$$r_{ui} = \sum_{(j,v) \,:\, j \in N(i),\ v \in N(u)} w_{jv} r_{jv} \tag{6}$$

where, $N(i)$ is the set of movies which have the highest similarity with movie i according to measure defined in Eq.4, analogously, $N(u)$ is the set of users v which have the highest similarity with user u according to measure defined in Eq.5, and w_{jv} is the weight assigned to the pair (j, v) respect to the pair (i, u). We tried different types of weights, such as

$$w_{j,v;i,u} = (S_{ij}^2 + S_{uv}^2)^{-1} \tag{7}$$

[1] in the user-based version when calculating the similarities function we would need to evaluate about $480,000 \times 480,000 \sim 2.3 \cdot 10^{11}$ values

i, j, k	Indexes for movies
u, v	Indexes for users
r_{ui}	rating of user u for movie i (real or predicted)
S_{ij}	measure of "similarity" between movies i and j
S_{uv}	measure of "similarity" between users u and v
$N(u)$	set of neighbor of user u, i.e. user which are similar to u
$N(i)$	set of neighbors of movie i, i.e. movie which are similar to i
$N(u;i)$	set of neighbor of user u which actually rated movie i
$N(i;u)$	set of neighbors of movie i which actually have been rated by user u

Table 1: Summary of Notations Used

This weight assignment codifies the intuition that the "further away" the two pairs (i, u), (j, v) are, the smaller the weight assigned to this interdependence should be. If fact, if we interpret S_{ij} as the "distance" between two movies, and S_{uv} as the "distance" between two users, then $w_{j,v;i,u}$ is inversely proportional to the square distance between the pairs (i, u) and (j, v).

We also tried another weight assignment which aims to express a similar intuition:

$$w_{j,v;i,u} = \Big(\frac{1}{S_{ij}S_{uv}}\Big)^{\epsilon} \tag{8}$$

where, $\epsilon > 1$ is an exponent that we tuned during our experiments.

Despite trying different assignment of weights with varying neighorhood sizes for users and movies, the above method does not seem to be able to yield good performance. In the experiments we were able to run the best value of RMSE reached was about 1.08. This may also be partially due to few attempts we performed to find the right tuning of parameters as well as the fact that prior to the computation of similarities we did not normalize our dataset to eliminate biases (discussed in [4]).

6.2 A Modified Version of the BellKor Algorithm

In this section we illustrate how we implemented the algorithm presented in [4]. In [4] the authors present part of the solution which won the progress prize. As aforementioned, the core of any nearest neighbor-based scheme is always how to define the neighborhood, and how compute the weights which provide the best estimate of the unobserved ratings. In order to identify the *closest neighbors* we first need to define a similarity function. In [4] the authors, after several tests with other functions, propose to use the following measure of similarity:

$$s_{ij} = \frac{|U(i,j)|}{\sum_{u \in U(i,j)}(r_{ui} - r_{uj})^2 + \alpha} \tag{9}$$

where, $U(i, j)$ is the set of users that rated both movie i and movie j, and parameter α is used to avoid a division by 0. We set $\alpha = 1$ in our execution. The values s_{ij} are pre-computed once, and stored in an binary format which allows access with the smallest possible latency. We note that the memory requirement to store only the values of s_{ij}, which are actually needed to precompute the neighborhood as it is proposed in [4], is about 1.2GB. The reason for that is that we store each entry as a float (4 bytes). The similarity Eq 9 is not as predictable as the cosine distance in regard to the range of values it takes (parameter $alpha$ also affects in that aspect), hence we were not able to store each similarity value s_{ij} in 1 byte (as described in [4]) without loosing considerable precision.

The values s_{ij} are then used to calculate the k-neighborhood for different values of k. Specifically, for every movie i, and user u, we calculate the set of the (at most) k movies which have higher similarity with movie j and are rated by user u. According to the notations defined in [4], we name this set $N(i; u)$. In order to predict ratings we now need to find the set of weights, w_{ij}, that provide the best estimate of unobserved ratings,

$$r_{ui} = \sum_{j \in N(i;u)} w_{ij} r_{uj}$$

We can formalize this problem as an optimization problem by first considering the hypothetical case in which all users but u rated both movie i and all its neighbors in $N(i; u)$. In this case, we can learn the interpolation weights w_{ij} by embedding this problem in a classical least squares problem:

$$\min_{w} \sum_{v \neq u} \left(r_{vi} - \sum_{j \in N(i;u)} w_{ij} r_{vj} \right)^2 \tag{10}$$

Differentiating the above equation we can find the optimal weight as solution of a linear system

$$Aw = b$$

where, A is a $K \times K$ matrix, such that

$$A_{jk} = \sum_{v \neq u} r_{vj} r_{vk}$$

and b is a vector of size K satisfying

$$b_j = \sum_{v \neq u} r_{vj} r_{vi}$$

For a sparse matrix, such as the Netflix matrix of user-movie, we can not perform the above calculation directly, but we can provide estimation of such quantities. In particular we use

$$\bar{A}_{jk} = \frac{\sum_{v \in U(j,k)} r_{vj} r_{vk}}{|U(j,k)|}$$

as an estimate of A, and a similar formula to estimate b. We point out that, despite the single matrix A_{jk} is quite small (in our experiments, K is about 15-30), we need to recalculate a new matrix for every prediction we want to perform. Thus, in same case we may have to calculate the values A_{jk} per every pair of movies. Finally, we can compute the optimal weights by solving the linear system

$$\bar{A}w = \bar{b}$$

which is, in our case, a system of about K equations (and an equal number of variables). In order to solve the linear system we implement ourself an iterative algorithm which is based on the Gradient Projection method. The details of the algorithm are described in Alg. 1.

Algorithm 1 SolveLinearSystem($A \in \mathbf{R}^{K \times K}, b \in \mathbf{R}^{K}, \epsilon \in R^{+}$)

while true **do**
  $r = b - Ax$
  if $||r|| < \epsilon$ **then**
    return x
  end if
  for i=1,...,k **do**
    if $x_i = 0$ and $r_i < 0$ **then**
      $r_i = 0$
    end if
  end for
  $\alpha = \frac{r^T r}{r^T A r}$
  for $i = 1, \ldots, k$ **do**
    if $r_i < 0$ **then**
      $\alpha = \min(\alpha, -\frac{x_i}{r_i})$
    end if
  end for
  $x = x + \alpha r$
end while

Despite the many improvements with respect to the previous algorithm, this method yields a relatively high RMSE similar to the first method in section 6.1. A simple modification, which can be seen as a form of data regularization, was needed to drastically boost up the performance of the our prediction system. To predict a rating r_{ui} we use:

	k=5	k=10	k=15	k=20	k=25	k=30	k=35	k=40
RMSE	0.981721	0.962698	0.960012	0.960932	0.96268	0.964587	0.966318	0.967928

Table 2: Results from execution with Eq. 11 for 100% of the probe dataset

$$r_{ui} = avg(u,i) + \frac{\sum_{j \in N(i,u)} s_{ij}(r_{uj} - avg(u,j))}{\sum_{j \in N(i,u)} s_{ij}} \tag{11}$$

$avg(u,i)$ is the *double average* for user u and movie i defined as :

$$avg(u,i) = \frac{\sum_{\forall i} r_{ui}/|r_{u*}| + \sum_{\forall u} r_{ui}/|r_{*i}|}{2} \tag{12}$$

where $|r_{u*}|$ is the number of movies voted by user u and $|r_{*i}|$ the number of users that voted movie i.

With this simple modification we were able to get RMSE as low as 0.96 in the probe data set. Table 2 contains the execution of this version with K, the neighborhood size, varying from 5 to 40. We should not here that K is the maximum number of neighbors. In some cases $N(i,u)$ might have less than K members in the set. We observe that increasing the number of neighbors to more than 20 in the items-based approach deteriorates the performance.

7 Conclusion

In this project we implemented three different variations of the k-nearest neighbors approaches for the Netflix competition, one of which is the outcome of this project. The source code (in C++) for all three versions including the precomputation steps is made available together with the report to reproduce the results obtained.

As future work, we would like to investigate in-depth the effects of data regularization of the initial dataset. As mentioned in [1] this may dramatical events in improving the final prediction. Same effects are intuitively clear, such as the fact that a certain user may tend to rate higher on average than another user, and this can be easily accounted by centering the ratings around the average user rating, that is: $r_{ui} \leftarrow r_{ui} - r_u$, where r_u is the average rating for user u. We did not perform any data normalization as a preprocessing step before similarity precomputation in this project, due to lack of time, and to limit our attention on the implementation of several variation of kNN methods. However, we deem this preprocessing phase will bring further improvements in our estimations as hinted from our last attempt in the previous section.

References

[1] Bell, R.M., Koren, Y. & Volinsky, C. (2007) "The BellKor Solution to the Netflix Prize", `http://www.netflixprize.com/assets/ProgressPrize2007_KorBell.pdf`

[2] Bell, R.M., Koren, Y. & Volinsky, C. (2007) "Modeling Relationship at Multiple Scales to Improve Accuracy of Large Recommender Systems" *Proc. 13th ACM SIGKDD International Conference on Knowledge and Data Dissemination*

[3] Netflix Dataset: `http://www.netflixprize.com/download`

[4] Bell, R.M., Koren, Y. (2007) "Improved Neighborhood Based Collaborative Filtering", *KDD 2007 Netflix Competition Workshop*

[5] Icefox Netflix Recommender's Framework `http://www.icefox.net/programs/?program=NetflixRecommenderFramework`

[6] Netflix Leaderboard: `htttp://netflixprize.com/leaderboard`