APAM E4990 Final Project

Building Recommendation System for Netflix

Siheng He

May 5, 2012

1. Introduction

The objective of this term project is to build an example recommendation system by applying machine learning techniques to real-world data. After major industry revolutions, the world has provided us tremendous amount of products and will continue to offer more. How to find out a few items that may interest someone, like music, movies, or books, among millions of product options is of essential importance in improving personal life standard or corporate profit. People may get suggestions from their friends who have similar taste on a specific category. As more and more choices become available, it is less practical to pick interesting items by asking a small group of people, since they may not be aware of all the choices, or asking a large group of people, since it may take long time. Fortunately, due to rapid development of web and information technology, it is now possible to collect information of large group of people on their choice. With the collaborative filtering algorithms, it searches in the information to find a small group of people that have similar taste to one person, looks at other things this group of people like and combine them to make recommendations for the person. Based on this idea, we could build a recommendation system and apply to real-world problem, for example, Netflix movie recommendation.

2. Data

2.1 Data acquisition

The Netflix dataset was constructed to support Netflix Prize and can be downloaded at a personal blog[1]. The size of the whole dataset is over 2 GB. The movie rating files contain over 100 million ratings from 480 thousand randomly-chosen, anonymous Netflix customers over 17 thousand movie titles, each rating file corresponds to one movie. The ratings are on a scale from 1 to 5 (integer) stars. The first line of each file contains the MovieID followed by a colon. Each subsequent line in the file corresponds to a rating from a customer and its date in the following format:

```
CustomerID, Rating, Date
```

As an example, the first two lines from mv_0000100.txt are shown below:

```
100:
2625420,1,2004-06-03
```

As a basic understanding of the data, Figure 1 shows the distribution of number of users a movie has, while Figure 2 shows the reverse side, i.e., the distribution of number of movies a user has watched.

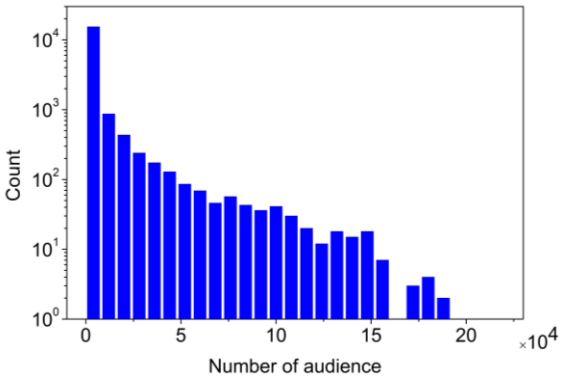


Figure 1 Distribution of movie popularity

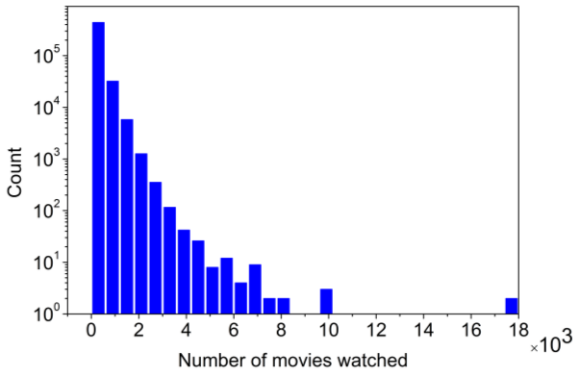


Figure 2 Distribution of user record

2.2 Data processing

The original dataset only provides training data (it does provide qualifying data without rating). In order to evaluate the performance of the recommendation system, the data in training_set folder is split into non-overlapping train and test folders, using function prepare in module prepare.

The original dataset has rating file for each movie. To get a user-centered rating file, the original rating files have to be reversed and group according to user. MapReduce technique is a perfectly suitable tool to achieve this goal. Amazon EMR (Elastic MapReduce)[2] and S3 (simple storage)[3] services provide an easy, user-friendly way to apply the technique. Upload the dataset (merge all the rating file into single one), user-defined mapper and reducer python scripts to S3, and set up a streaming job flow in EMR management console like following:

Input Location: s3n://pekingcolumbia/data/trainfile
Output Location: s3n://pekingcolumbia/ouput
Mapper: s3n://pekingcolumbia/program/mapper.py
Reducer: s3n://pekingcolumbia/program/reducer.py

then we will get the results from S3 output folder. The resulting 26 ‘part-xxxxx’ files must be downloaded from above location and put in the current project folder to ensure normal functionality of the main python script.

3. Recommendation System

3.1 Matching movies

This recommendation system is based on item-based filtering. First we find similar movies. If a large fraction of common users give the same or close ratings to two movies, the two movies are highly similar. To quantify the similarity between two movies, users who watch both are picked out. The difference of ratings given by a user to the two movies is squared, and summed over all these users. Obviously this is the square of Euclidean distance d in user space. Similar movies are closer, hence larger d indicates less similarity. If two movies don't have common users, their similarity is defined to be 0; otherwise similarity is defined as

$$\text{similarity} = \frac{1}{1+d}$$

the addition of 1 in denominator avoids infinite value when two movies have exactly the same ratings for all common users, which is unlikely to happen. The similarity can be calculated by sim_distance(movie1,movie2) function in recommendation module.

3.2 Rating movies

For any (user,movie) pair in testing dataset, the recommendation system aims to predict how the user is going to rate the movie. The function getRate(user, movie, similarity) in recommendation module first searches user history to find a list of movies that the user has rated in the past (i.e., in the training dataset). The similarity between the argument movie and each item in the list is calculated using the passed-in argument method similarity. The ratings the user gives to each item is weighted by the corresponding similarity score and then summed, which is further normalized by dividing by the sum of similarity scores. The resulting number is the prediction of numbers of stars.

3.3 Performance

3.3.1 Time cost

Due to the huge size of the dataset, the recommendation prediction time is on average 30 minutes, much longer than smaller dataset like MovieLens Dataset[4]. The distribution of runtime is plotted in Figure 3. Runtime varies between 2 minutes and 3 hours, depending on the size of user watching history and movie popularity. A deeper investigation of runtime shows that the majority of time is spent on calculating the similarity score. The function used for similarity calculation reads two movie training files from hard disk and pick out all the common users and associated ratings, a process that could take several minutes for large files such as mv_0000571.txt (3.18 MB).

3.3.2 Accuracy

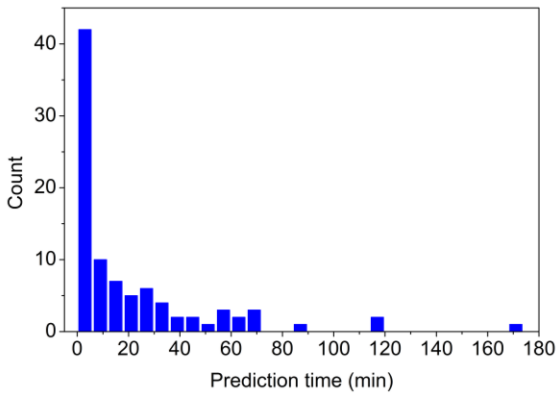


Figure 3 Distribution of prediction time cost

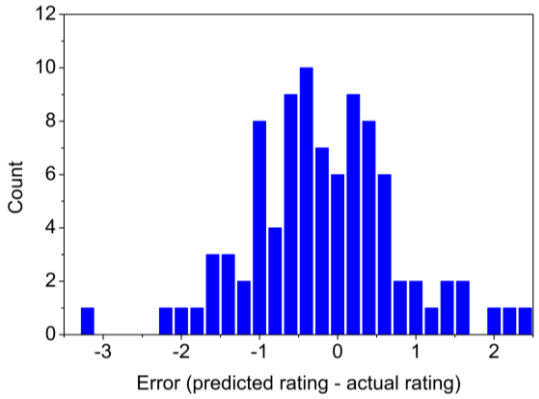
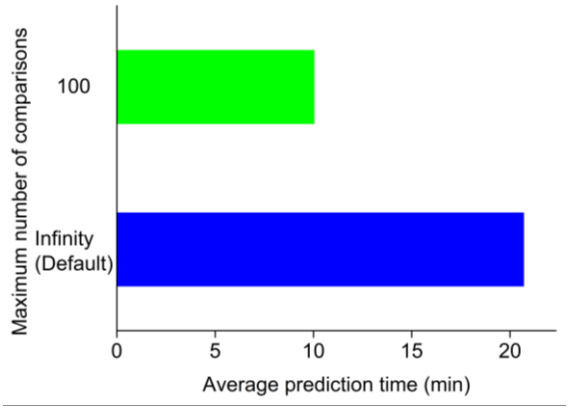
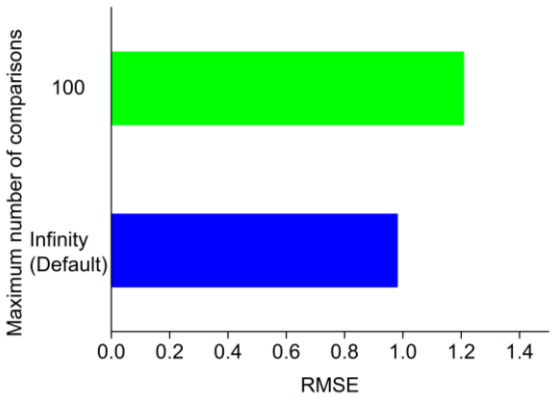


Figure 4 Distribution of prediction error



(a) Time cost comparison



(b) Accuracy comparison

Figure 5 Performance at different complexity level

Due to the long runtime and limitation of computing power, the recommendation system only predict ratings of 91 users for 6 randomly chosen movies. The accuracy is quantified by root-mean-square error (RMSE), whose formula is

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(x_{prediction,i} - x_{actual,i})}{n}}$$

Based on the test, the RMSE of this recommendation system is 0.9812, therefore the accuracy is reasonably well for such a simple system. Winner of Netflix Grand prize achieved a lower RMSE, 0.8567, using more advanced techniques[5].

3.3.3 Complexity control

In this recommendation system, by default the movie to be rated by the user is compared to each of the movies the user has seen. For users who have rated a lot of movies in the past, this similarity comparison process could be cumbersome and time-consuming.

However, the more movies the user has watched, the clearer his flavor for movies is, hence the accuracy of prediction may increase. As an example test, the recommendation system is tested on the same test dataset as in the default mode, but the maximum number of comparisons is limited to 100. The performance of the two modes is illustrated in Figure 5.

4. Conclusion and Discussion

Although the recommendation system here use the simple Euclidean distance as a measure of similarity of two items, there are actually many other metrics to measure similarity, such as Pearson correlation, Jaccard coefficient or Manhattan distance. Choice of methods will depend on specific applications. Jaccard coefficient measures the ratio of intersection over the union of two datasets, so it may lose the detail information in five-star scale rating and is not appropriate for movie recommendation. It may be better to be used in all-or-none scale rating systems, like bookmarks recommendation.

This recommendation system uses item-based filtering strategy. The other alternative is user-based filtering. Item-based filtering usually performs better than user-based filtering in sparse datasets, but roughly both perform equally well in dense datasets[6]. Movie flavor datasets like Netflix dataset are generally dense since most users have watched some same movies and vice versa. In addition, often there are much more people than movies, and the pool of users is changing more rapidly, it takes more effort to find similar users for a specific user. On the other side, movies data are relatively stable. Moreover, if we need to provide recommendations for large number of users and movies, which many website deals with every day, the recommendation system spend long time preparing the similarity data between all movies and stores the data, the time of recommendation will significantly reduce.

In conclusion, this project utilizes simple techniques, scales to practical real problems and works reasonably well. It’s also more interpretable, which makes a good machine learning practice.

Reference

1. pinwei. *netflix dataset 下载*. Available from: http://www.lifecrunch.biz/archives/207.
2. Amazon. *Amazon Elastic MapReduce Getting Started Guide*. Available from: http://docs.amazonwebservices.com/ElasticMapReduce/latest/GettingStartedGuide/Welcome.html?r=3298.
3. Amazon. *Amazon Simple Storage Service Getting Started Guide*. Available from: http://docs.amazonwebservices.com/AmazonS3/latest/gsg/GetStartedWithS3.html?r=7608.
4. University of Minnesota, G.R., *MovieLens Data Sets*, 2011.
5. Koren, Y. *The BellKor Solution to the Netflix Grand Prize*. 2009.
6. Segaran, T., *Programming Collective Intelligence*. 2007: O'Reilly.
7. Hofman, J. *APAM E4990 Data-Driven Modeling Class Notes*. 2012.