Recommendation system report

Puyang Ma

May 2017

1 Introduction

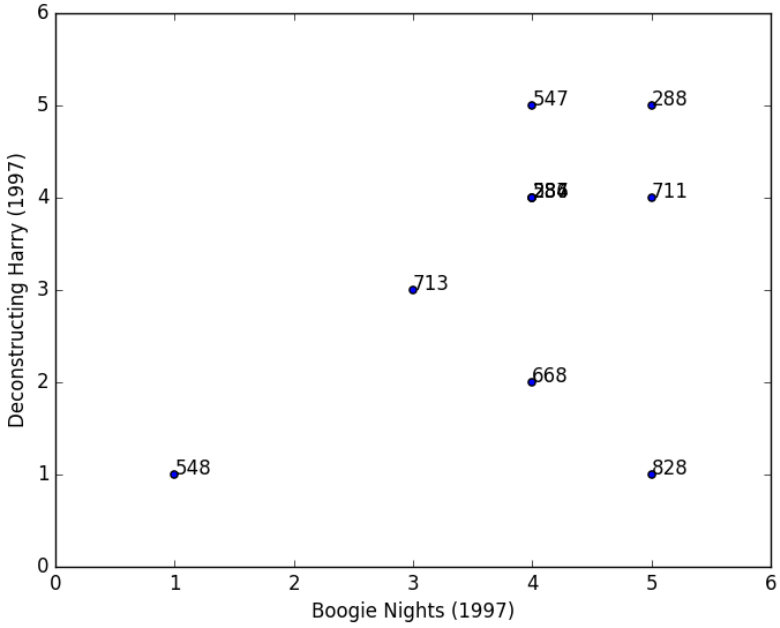
With brimming data available online in different fields, it is possible to discover patterns and develop useful applications from this rich resource. One of the most striking example of the power of data science is recommendation system (recommender). Companies like Amazon, Netflix and even Target have long since utilized these systems for individualized product promotion. As suggested by the name, a recommender uses information about users, such as their purchase history, ratings for specific products, and demographics to predict the relevance of a particular item. Such predictions can be quite accurate: There was a piece of news about Target knowing the pregnancy of a teen months before her relatives did. And that foresight came from recommender based on purchasing patterns of pregnant women. Not only do companies need such systems for efficient promotion, customers/users also need personalized recommendation in face of gigantic number of choices. These systems often serve two purposes, one is *annotation in context*, which predicts the rating the user would give to an item based on his/her search and browsing history; the other is *find good items*, which recommends a list of items it considers most relevant to the user. Technique wise, there are in general two types of recommenders, the content-based filtering and collaborative filtering. An example for the former one would be retrieving text documents based on a set of keywords present in the user profile. Limitations of content-based filtering are its ability to distinguish between high and low quality texts that both include those keywords, and its ability to analyze multimedia content where machine perceptions differ from that of users. Due to these limitations, I chose to use collaborative filtering to build my recommender. It is basically the same process as seeking suggestions from friends and noting choices of people who are similar to you to inform your own decisions, but the process will yield more accurate results if it is enhanced by abundant data and a quantitative measure of similarity as in collaborative filtering.

2 Data exploration

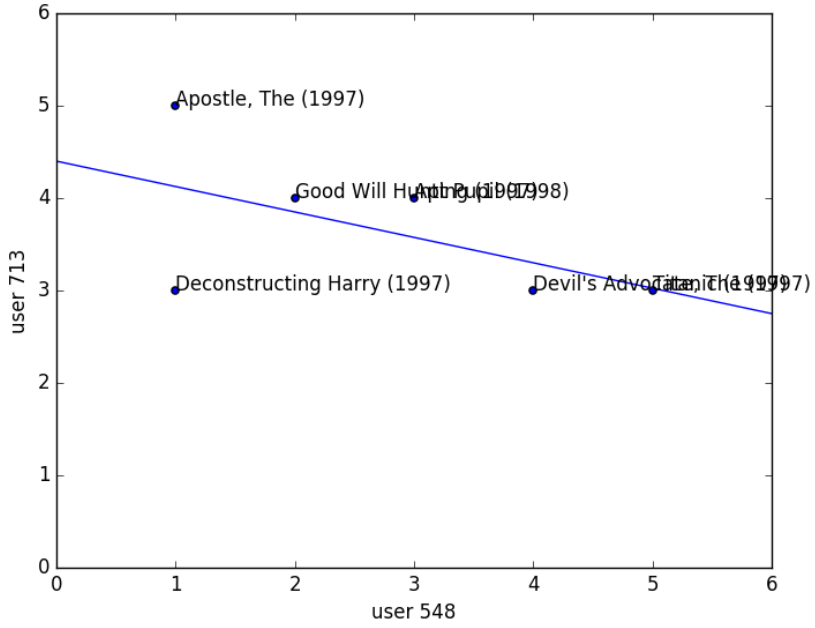
The project is a movie recommender, and is based on grouplens 100k data set. The data set consists of 100k ratings (1-5) from 943 users on 1682 movies.

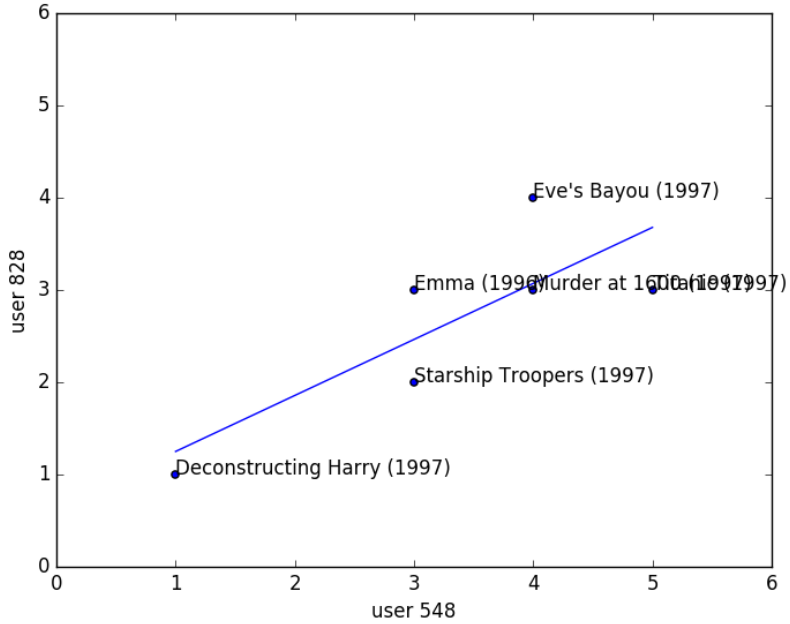
Each user has rated at least 20 movies. And the data was collected through the MovieLens web site (movielens.umn.edu) during the seven month period from Sept 19th, 1997 through April 22nd, 1998. I started by exploring the data, observing relationships among users, and testing my assumptions against these observations. One of my assumption was that if the Euclidean distance of ratings between two users is small, then they can serve as good reference for each other (neighbor). However, several figures I plotted show examples where the Euclidean distance can be misleading.

Figure 1: movie as axis



In Fig1, user548 and user713 has the smallest euclidean distance of $\sqrt{8}$, while user548 and user828 has much larger euclidean distance of $\sqrt{16}$. Based on the Euclidean distance, I expected user548 to be closer with user713 than user828.





However, the plot of "user548 vs user713" shows a best fitting line of negative slope while the plot of "user 548 vs user828" shows a best fitting line of slope

near 1. In sum, contrary to what the euclidean distance suggests, user548 has rating patterns more similar with user828 than with user713. Comparing the previous two plots, it is clear that although the rating difference in absolute value is smaller between user548 and user713, they actually have opposite tastes in movies. Euclidean distance hides this piece of information.

3 Algorithm

3.1 Distance Metrics

In terms of capturing similar rating patterns between users, Pearson coefficient is a more suitable metric than Euclidean distance. Pearson correlation, which measures linear correlation between two variables,is by definition the covariance over the product of their standard deviations. For a sample, it can be calculated by $r = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y}}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$. Although python has its built in Pearson correlation formula, I wrote it myself to make sure the case of zero denominator is dealt with properly. In my context, I decide that if ratings of a user has zero variance, then the user does not correlate with other users (Pearson coefficient is zero). There are also other kinds of distance metrics for similarity measurements: cosine similarity, for example, measures how similar the orientations of two non-zero vectors are. If they have the same orientation, the angle between them is zero, thus the cosine is one. When the two vectors are perpendicular, the cosine is zero. The centered cosine similarity, where vectors are normalized by subtracting the vector mean, is equivalent to the Pearson coefficient.

3.2 Recommendation Logics

To recommend a person items from the set of 1682 movies, the system uses a distance metric (e.g. Pearson) to calculate the similarity between the person and the rest of the users, then selects a subset of users according to their similarity with the person. The subset can be chosen either by a distance threshold or by max number of similar users (neighbors). Finally, using neighbors' ratings the system predicts ratings by the person on movies he/she has not seen. The prediction value of a movie is $\frac{\sum(rating_i)(correlation_i)}{\sum correlation_i}$. It is necessary to include the denominator as a normalization since some movies get more ratings than others, thereby having higher predicted value.

```
for each user other than u_a ∈ data do
    if user != u_a then
        calculate distance btw user and u_a end
        for each mv rated by user do
            if mv not rated by u_a and distance ≥ threshold then
                calculate prediction mv;
            end
        end
    end
    sort prediction in descending order;
    return top N mv
```

Algorithm 1: personal recommender pseudo code

Another way to calculate the prediction is by Z-score normalization, which is a refinement of the crude weighting by correlation. The Z-score normalization takes into account different types of users: users who only rate good items, those who are harsh raters, others who are more lenient, etc. So ratings of users have different distributions, and normalization needs to include mean and standard deviation of each user: $P_a = \bar{v_a} + \sigma_a \frac{\sum u \in Neigh_a [\frac{v_(ui) - \bar{v_u}}{\sigma - u} s(a,u)]}{\sum u \in Neigh_a s(a,u)}$ The Z-score weight essentially removes the rating pattern of a neighbor to find a prediction for the specific distribution of the active user.

3.3 Group Recommendation

After getting a personal recommender, it is then easy to use it to implement a group recommendation: for a group of people, personal recommender suggests for each person a list of potential favorite movies, and the collection of all lists form the pool of movies to recommend from. The system can compute a group rating for each movie in the pool by summing each person's score weighted by his/her importance in the final decision. For example, adults who accompany children for a movie may be willing to give more weight to child's preference.

3.4 Item-Based Recommendation

The run time for my personal recommendation system is less than a second for 100k data and is about 40 seconds for 1M data. However, this is the performance in a static data set. Imagine running the system online where the number of users grow exponentially and each user often rate new items, then the system needs to constantly updates similarity scores between users, and it could take hours to run the system. On websites like Amazon, the number of items does not grow as fast as that of customers; therefore, similarity between items tends to be more static than the similarity between users. For this reason, I also implemented an item-based recommender using the same logic as that of the user-based system, with the only difference in that the system now recommends items most relevant to an item rather than a user.

4 Conclusion

In the process of building this side project, I learned to program in Python, applied statisitcs to solve practical problems, and read about machine learning algorithms such as support vector machine and k-nearest neighbor. I also identified several areas of my system that needs further improvements: The personal recommender that includes Z-score normalization currently has a run time of about 20 minutes for the 100k dataset, which is far less than ideal. I plan to work on improving the run time significantly. Moreover, according to Sarwar et al. [2001], by far the most accurate distance metric for item-based algorithms is adjusted cosine similarity and I plan to implement it in my system as well.