Demystifying movie ratings

224W Project Report

Amritha Raghunath (amrithar@stanford.edu)
Vignesh Ganapathi Subramanian (vigansub@stanford.edu)

9 December, 2014

Introduction

The past decade or so has seen a massive explosion in the amount of information and data that has been passing through social networks. These contemporary networks carry millions of nodes, and billions of edges, with zetabytes of information being passed around on a minute to minute basis. Analyzing this information to observe patterns is a very vital cog in the process of feedback, where these networks learn more about the attributes of their users. Often, these attributes are obtained from basic mathematical analysis of the information we observe.

User recommendation systems are one of the by-products of obtaining attributes from observables in a social network. Recommendation systems are used in almost every online service today, be it product sales, such as in Amazon, or choice making, in Yelp, movie recommendations in IMDb, book recommendations on Goodreads, question recommentdations on Stack Overflow or Quora or even friend recommendations in Facebook or Twitter. A lot of these times, recommendation systems do not account for the noisy ratings by other users. For example, when many computer scientists decide to independently read a popular sports book, it is possible that for some other person searching a book on social network analysis, a recommendation to buy the latest year book of a popular soccer club pops up.

This is a problem with a lot of movie ratings we observe on user rated movie websites such as IMDb. In this project, we use ideas from community networks to obtain the "true" movie ratings of a movie, by trying to get rid of noisy user ratings. In [1], the authors introduce the notion of a network community profile plot, where the quality of the best possible community in the network is measured, by finding communities of various sizes that minimize the community values.

The property of modularity, which measures if a particular division of a network into communities is a good division or not, is maximized in [2]. This property gives a measure of the "distance" of a particular community assignment, to a structure in which edges are randomly distributed. The maximization is done in a time optimal framework where three data structures are maintained. The first data structure is a sparse matrix containing current incremental modularity values between different communities, Each row of this matrix is stored as both a balanced binary tree (for inserting values), and a max-heap, to find in constant time the maximum element of the row. The second data structure contains the maximum of each row, with the row and column indices, and the third data structure contains the fraction of ends of edges in each community.

The paper uses update rules to merge small communities to form larger ones, up until the point where the modularity starts decreasing, using a greedy algorithm. This point at which the modularity hits the peak would be the optimal community distribution. This modularity expression is given by

$$Q = \sum_i (e_{ii} - a_i^2) \tag{1}$$

where e_{ij} refers to the fraction of edges that connect vertices in community i to those in community j, and a_i refers to the fraction of ends of edges that are connected to vertices in community i.

In [3], there is further discussion of modularity and it's uses in finding community networks. This paper begins with the schematically well defined example of a ring of cliques, connected to each other by a minimal number of links. Then the paper discusses a very general model of more than three modules, where it models the interactions between two individual models and their interaction with the rest of the graph structure. The core of this paper though, is the idea of the resolution limit of modularity, which depends on the interconnectedness between pairs of communities and can reach the order of size of the entire network. Here the authors attempt to maximize the modularity, and resolve modules by modularity optimization, and show that this cannot be achieved. They also illustrate how entire modules of large sizes go unnoticed since they share a lot of links with the rest of the network

Most community detection algorithms are for networks where all the nodes are of the same type. The problem of community detection becomes slightly different when it comes to bipartite graphs. In [5], we have two different kinds of nodes, blue and red say. For example. in the movie-rating networks, we have movie vertices and user vertices. This paper proposes to use Barber's Bipartite Modularity, an extension of modularity for the bipartite case. Based on this definition they combine two different algorithms for community detection.

In the first algorithm, BRIM's, we initially start with a division of of one type of nodes say red. We then induce the division of the blue nodes. Next we try to induce the division of red nodes from the division of blue nodes. We iterate till we reach a local maximum.

A bad initialization in BRIM's can result in a poor division. For this reason a second algorthim, Label Propogation, is used to get an initial division. Here we start by assigning a unique label to every node. We then propogate the label. At every step, each node updates it's label to the maximal label of its neighbors. Ties are broken arbitrariy. This helps in getting a quick initial division.

One other method used to build a recommender system, in [6] is to use the belief propogation technique. The message passing algorithm for this technique uses a probabilistic model. Nevertheless, the idea of passing information across a bipartite graph to all other neighbors of a node's neighbor is extremely potent in standardizing the information across the system.

In our project we plan to use belief propogation, to evaluate a particular movie rating network. We first build a network, which is bipartite in movies and users, and try to obtain a quantity we define as the user offset. Based on these user offset values and original ratings given by a user to a movie, we try to give the movie a rating which we compare against metacritic ratings, and plan to present our results eventually.

Problem Statement

In this project we will try to predict the "true" ratings for movies. Our assumption in this endeavor is that "true" ratings are sufficiently approximated by Metacritic scores of movies. The aim of this project is to use properties on the graph generated by the Amazon users and movie ratings, and obtain the value of these "true" ratings, using data from Metacritic for training and testing.

Approach

Data

We are using the Amazon movie reviews dataset to create a bipartite graph of users and movies. This data is already readily available from the SNAP website. The dataset we have used contains 1540 movies in the training set, with 109694 users and 552980 movie ratings.The test set contains 200 movies, 24722 users (who obviously have all been part of the users in the training set), and 36934 ratings (from the Amazon pull-up). For consistency purposes, and to eliminate outliers, we extracted movie ratings from only those movies on Amazon, which had more than 50 reviewers.

We consider metacritic ratings as the absolute rating for evaluation purposes. We scrape their site for ratings of movies we're interested in evaluating.

Baseline Model

As a baseline approach, we have attempted to find reliable users without any belief-propogation. For every movie we have picked out the best $k-\%$ modal values of a movie's ratings, where k is a threshold value. We then compare the various k weighted rating values against the true movie ratings from Metacritic. It turns out that using the complete set works best in this case, and so we observe that without using properties of social networks, the movie ratings do not improve in any manner, since the method of obtaining relevant users is by computing the modal rating and this is not an accurate method to compute relevant users since a lot of the ratings are skewed naturally towards the highest possible ranking and so clusters are invariantly going to be around this highest ranking.

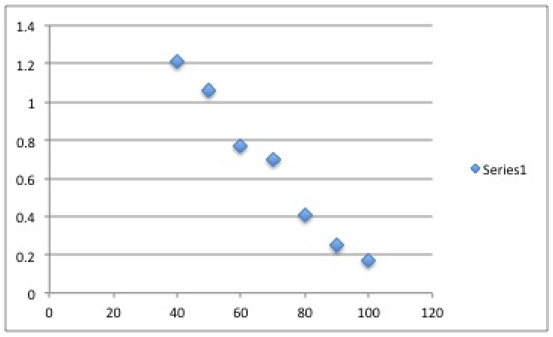

Figure 1: This plot contains the change in mean-square error with threshold % values

As discussed earlier, the ratings of most movies under this baseline scheme seems to be best for the entire dataset of users, as compared to a certain set of users. We present the predicted ratings of a few movies, along with the original rating. The movies whose results we present are The Count of Monte Cristo, Super 8, the Last Samurai, and Saw. The corresponding results (predicted followed by true rating in brackets). are $4.06(4.2), 3.60(3.65), 4.12(3.85)$, and $3.68(3.6)$ respectively.

We have scraped Metacritic for a list of 200 movies which are available also in Amazon reviews, around 20 movies from each decade at random, over the last 100 years, in which case the movies we take are randomly spread across timelines. These movies would be a fair source of error comparison, since across time frames we would be able to standardize ratings.

Note that all the work done so far does not make use of any of the social network properties, and any use of network properties can only make results better by making use of the relationships between different reviews by the same user and so on.

Translation Fit Model

In Figure 2, we have plotted the Metacritic scores of movies, and the corresponding Amazon averages of the same movies. Since there is an obvious disparity between the two but there is a visual similarity between them, we try to translate the Amazon data to fit the Metacritic rating more accurately.

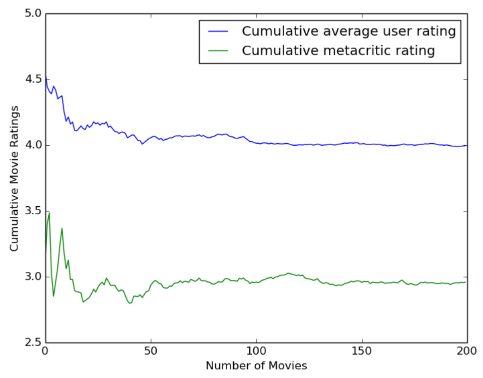

Figure 2: This plot contains the true Metacritic rating and average rating of movies based on Amazon user ratings

In this model, given the set of user ratings for each movie, we compute the average movie ratings for each movie, and try to find a constant c, by which we could plainly translate the average movie ratings on Amazon, to get closest as possible to our Metacritic rating (closest in a least square sense). So assuming x_i are the Amazon average movie ratings, and y_i are the corresponding Metacritic ratings for movie i, then we try to find c such that

$$E(c) = \sum_{i=1}^{N} (x_i - c - y_i)^2$$

is minimized. The $c = c^*$ value doing this minimization is given by

$$c^* = \frac{1}{N} \sum_{i=1}^{N} (x_i - y_i).$$

We now translate the Amazon data by c^*. Figure 3 shows the cumulative mean squared error plot of the same.

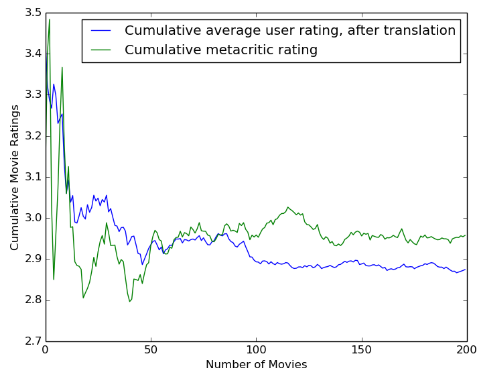

Figure 3: This plot contains the true Metacritic rating and translated average rating of movies based on Amazon user ratings

Belief Propogation Model

In Figure 4, we have a bipartite graph G on the set of users and movies. Each user reviews a number of movies, with a rating value between 0 and 5, and each movie is reviewed by a number of users. The graph is bipartite by definition. The two parts of the graph are the users and the movies. The edge between a user node and a movie node refers to the fact that the user corresponding to the user node has rated the movie corresponding to the movie node in Amazon. This edge in Stage 1 corresponds to obtaining user ratings of movies, and in Stage 2, corresponds to returning the offsets from movies to the users. Here we use an idea similar to that of belief propogation. This algorithm is implemented in two stages. In the first stage, each movie node, containing the value of an oracle movie rating, gets the rating each user gives it, and sends back the offset of the user from the rating. The second stage corresponds to the user nodes computing their average offset, and then offsetting their original rating by the computed average offset. This process is repeated again, through Stage 1, until convergence.

Stage 1

In the given graph, we initially have each movie receiving the set of reviews by the set of users reviewing the particular movie. The blue nodes are the transmitting nodes, and the green nodes are the receiver nodes. This would mean incoming information through directed incoming edges of the movies. The movie nodes compute the offset of each user's rating from the oracle movie rating.

Stage 2

This offset is transported back to each user. For example in the above figure, if the rating of movie 1 by user 1 is x_{11}, user 2 is x_{12}, and user 5 is x_{15}, then if the oracle value for movie 1 is o_1, then the offsets for the three users are $x_{11} - o_1$, $x_{12} - o_1$, and $x_{15} - o_1$. Now, let this offset suggested from movie i to user j be y_{ij}. The average offset encountered by user i is $p_i = \frac{1}{|K_i|} \sum_{j:j \in K_i} y_{ji}$, where $K_i = \{j : ji \in G\}$. The modified rating x_{ij} now is given by $x_{ij} - p_i$. This process is repeated till $p = [p_1, p_2, ..., p_n]$ has a norm that is sufficiently close to 0. The offset for user i is just the sum of the offsets p_i at every step of the iteration.

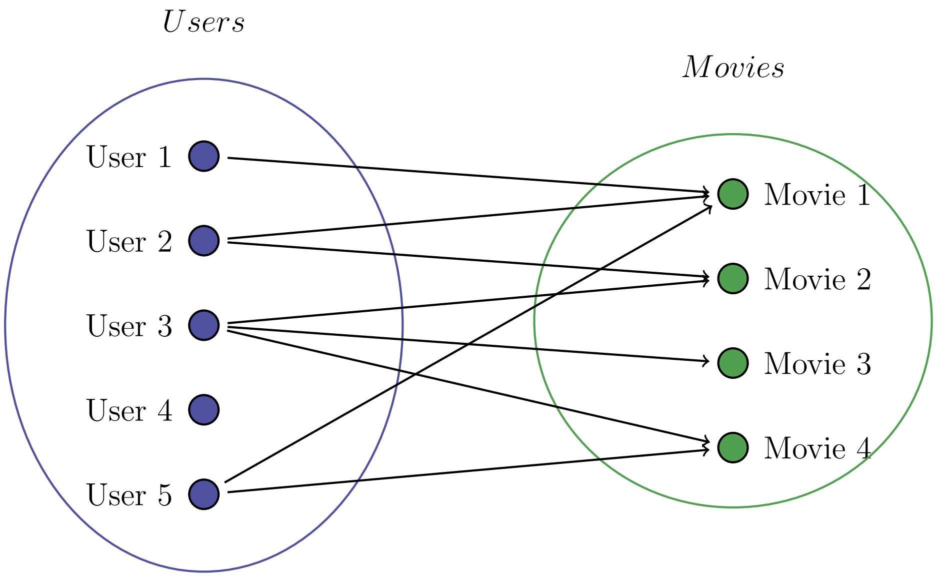

Figure 4: Stage 1

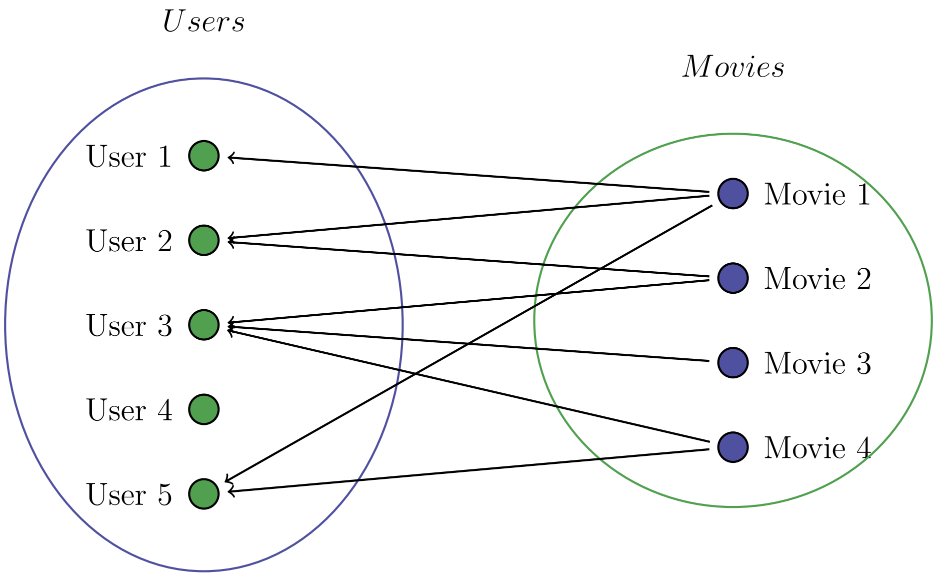

Figure 5: Stage 2

Oracle values

We shall now discuss how we to obtain the oracle values for each movie. One naive way of obtaining oracle values would be just plainly obtaining the average of all the input values x_{ij} a movie j receives from users i who reviewed it. Our main model consists of using the metacritic value of the movie as an oracle. In this case, the entire training set of metacritic data is used to compute the per-user offsets. These offsets are later used to compute the average "true" rating of a movie, given the ratings various users gave it.

Evaluation

We will use values from Metacritic as the golden set. However since Metacritic ratings are on a scale of $1-100$, we will want to look into methods of normalizing it into the 1-5 scale. These normalized values can be used to evaluate the system.

We can use a root mean square error metric, with respect to the normalized values. This would mean, we would like to reduce the root mean square difference between the normalized Metacritic rating, and our obtained rating based on Amazon user ratings. We would like to observe and find out if belief propogation helps in eliminating noisy ratings, by offsetting their effects.

Our error metric would be the simple mean square error given by $\sum_{i=1}^{N}(p_i-m_i)^2$, where m_i refers to the Metacritic rating on 10 and p_i refers to the predicted movie rating based on our available Amazon movie rating

We develop a comparison setup by first evaluating the translation fit model on the ratings. This is equivalent to a linear regression model, and therefore, this would be equivalent to comparing our belief propogation learning to a linear regression based model. This is expected to not perform as well as the belief propogation method used to learn the Metacritic rating.

The next benchmark of evaluation would be against the ratings obtained using the naive oracle method, where the oracle is determined by the average of all users having rated a particular movie. This benchmark effectively computes the "true" rating of a movie, without learning from the Metacritic ratings. In the other benchmark though, we have learning done, even though it is plain linear regression, done from the original Metacritic values. We would evaluate our model against all these benchmarks and try to determine how well it performs.

Results

Our aim is to get as close as possible to the Metacritic rating by eliminating noisy Amazon user ratings. This working out would lead to an extremely authentic and simple way of obtaining "true" movie ratings given a bunch of users and their ratings for movies. We try to evaluate our aim by using methods we discussed in our previous section.

We first plot the histogram of the absolute difference between the original Metacritic ratings, and predicted true ratings using our model. This is seen below in Figure 6.

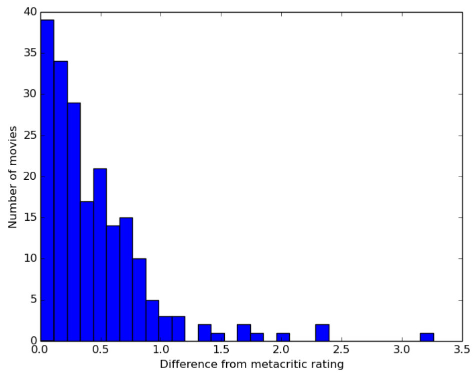

Figure 6: This plot contains the histogram of the movies based on their difference from Metacritic ratings, binned into 30 bins

Now, we also plot the cumulative rating of our model, with the number of movies tested upon in reffinalplot. This is compared to the previous two benchmarks as well. Here, very clearly we can see that the belief propogation model gives values very close to the original Metacritic rating, while the translation fit which is also close to the Metacritic rating does not converge to the same cumulative average with increase in the number of movies, while the belief propogation does. The belief propogation using just self-belief does not perform too well, and is way off the mark with respect to the rating.

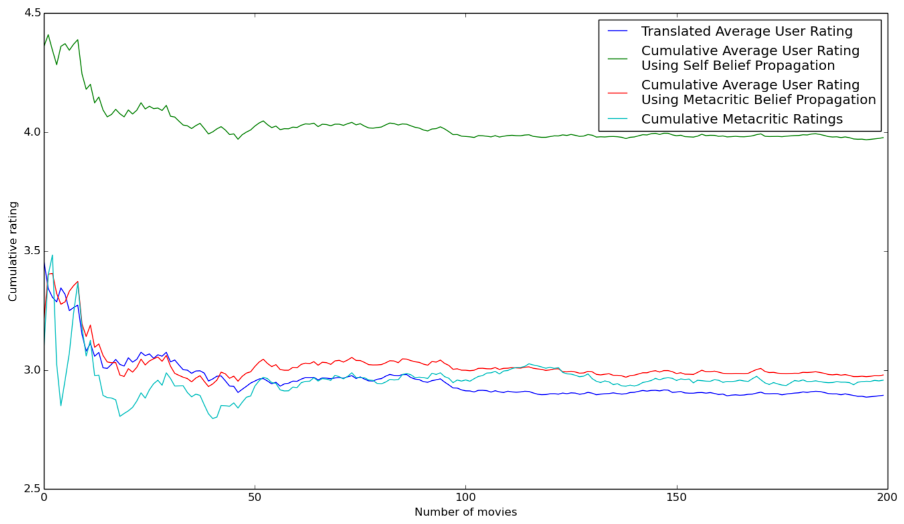

Figure 7: This plot contains the cumulative average ratings of the belief propogation model with the naive oracle, the belief propogation model with the Metacritic scores as Oracle and the Translation fit model, along with the original Metacritic ratings

The root mean-square error for the belief propogation method using Metacritic is 0.07, while that using Translated fit is 0.11. The RMS error for the belief propogation method without using Metacritic training though is 1.97, in accordance with the graphs obtained.

References

[1] J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Statistical Properties of Community Structure in Large Social and Information Networks. In Proc. WWW, 2008.
[2] A. Clauset, M.E.J. Newman, C. Moore. Finding community structure in very large networks. Phys. Rev. E 70, 066111, 2004
[3] S. Fortunato, S. Barthelemy. Resolution limit in community detection. Proc. Natl. Acad. Sci., 2007.
[4] X. Liu and T.Murata, Community detection in large-scale bipartite networks, Proc. of the 2009 IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology (WI-IAT 09), Vol.1, pp. 50-57, September 2009.
[5] J. Reichardt, S. Bornholdt. Statistical Mechanics of Community Detection., Phys. Rev. E 74 016110, 2006.
[6] E.Ayday, F. Fekri, "A belief propagation based recommender system for online services", Procs of the fourth ACM conference on Recommender systems, RecSys '10, Pages 217-220

Individual Contributions

All work was done jointly by both members of the team.