Board Game Recommendation

Sam Helmich

March 1, 2015

1 Introduction

Board gaming, while being very popular among children, is a niche hobby beyond adolescence. Everyone may have grown up with Monopoly, but most people may have never heard of Citadels or Twilight Imperium. And as such, board gaming is a very difficult hobby to break into. "Gateway Games" such as Settlers of Catan or Ticket to Ride offer a gradual transition between the ultra-popular and mechanics-lite games like monopoly, to more rules-heavy games like Puerto Rico. For most people, the difficult part of the transition is actually finding new games to play. New games are often expensive for people who may be apprehensive about trying new games (a copy of Settlers of Catan goes for about $40 on Amazon). For this reason, a method to recommend games to people would be invaluable.

A few methods of varying usefulness already exist. Boardgamegeek.com, a popular site among frequent board gamers, has a recommendation system that allows users to find a particular game they like, and then from there it will recommend more games that a person may like. A pretty thorough analysis of it (and its issues) can be found at `http://boardgamegeek.com/wiki/page/Game_Recommendation_Algorithm`. The bigget issue is that it cannot be tailored to an individual. You, an individual user, may have different preferences than the userbase average, and it might be more useful to tailor recommendations to the individual.

Recommendations for the individual aren't easy to come by. There are a few automated systems out there (such as the boargamerecommender bot on reddit), but for the most part they are time intensive and often lack interactivity or any sort of information that would be useful to the user beyond a few single games that it might recommend.

In this paper, we seek to build a system the will recommend board games to users based on their own recommendations as well as the recommendations of others. This system will offer the user not only recommendations for a very large number of games, but it will also offer insights to the user about what sorts of games they like, not only scores for particular games.

2 Data Collection

For data, we turn to `boardgamegeek.com`, a website that holds information, ratings, and reviews for over 75,000 games. Users can create an account and then rate games they've played as well as keep an inventory of games they own. All of this information is accessable and scrapable, and will be the data that drives our analysis.

There are two types of particular information that we will be interested in: user ratings of the games and characteristics about the game (referred to as classifiers in the modeling section). The characteristics we're interested in fall into these categories as outlined in table 1

Just to make this flow better - make the table below an actual table in the table environment. Give it a caption and refer to it in the text ... I've just implemented it for you

Category	This covers the broad categories a game might fall into (ex.: Humor, Puzzle, Sports, etc.)
Family	Some games a naturally part of a "family" that share a name or common element (ex.: Animals:Bats, Ancient Wars Series, Hello Kitty)
Mechanic	This is what mechanisms the game uses (ex.: Dice Rolling, Card Drafting, Worker Placement)
Subdomain	More general than Category, but loosely bins games (ex.: Family Games, Strategy Games, etc.)

Table 1: Overview of the relevant characteristics of board games considered in this project.

The ratings for each game are on a scale from 1 to 10, with a score of 1 indicating a really bad game and a score of 10 being the best possible. Each of the ratings are paired with a unique user identification number as well as a game number. Combined with the classifiers data, we can provide very specific ratings for each individual user.

Give an example of how the data that you have scraped looks like.

Aren't you continuously scraping? You should mention this at this point, and also talk a bit about how you have implemented the scraping.

3 Model

3.1 Model Selection

To begin implementing such a model, we can look to two papers on one of the most famous recommender systems of all: Netflix. In 2006, Netflix offered a $1 Million prize for a improvement on their algorithm for predicting user rating on movies they had not seen. In 2008, two teams: BigChaos and BellKor (the eventual winner), published papers on their methodology. In the end, they ended up taking something of a shotgun approach, using a linear blend of many different models to come up with a single super-model.

There are a few things that make sense to model first. Certain games are more popular than other games, and certain people will naturally be more or less generous than others in their ratings. These two effects are very important to measure, as the first will allow us to provide estimates for people that have rated no games (and therefore we know nothing about), and the second will allow us to fine tune estimates for people who have rated many games. We will implement these in the same way that BigChaos implemented the Global Effects model (page 6, BigChaos paper). While they implemented 14 steps, we will only use the 2, due to the complexity of the calculations required and the time with which it is practical to fit a model.

After we fit the general effects, we wish to model the remaining residuals, ideally in some way that gives us informative numeric summaries about the individual users in the process. In addition, we'd also like to compare a users ratings to other users, and hopefully use the behavior of simiar users to predict new ratings for someone. A tempting first step is to, for a particular game, find other people who have rated that game who have rated games that you've rated. Then, you could find people who rated games similarly to you, and use their ratings of the new game as an estimate. This often doesnt work, as sometimes the number of people who have rated the same games as you can be relatively low, if not 0 (show a graph for this part).

To get around the problem of low crossover in game ratings, we turn to the characteristics of the games themselves. This kills two birds with one stone. We'll create, for each user, a rating for each game mechanic and genre. Then, since there are so many mechanics and genres of game, a player only needs to rate a small, albeit diverse set of games before we can correlate that player's habits with other players, even if they've never played any of the same games! Then we can use those to filter rating results, as we'll know what board game mechanics that a particular player will enjoy.

We now have a very simple two step model that will allow us to predict ratings for new board games that a person may not have even played! Also, it allows us to rate games that nobody has played before: We can use the overall mean as a starting point, then for an individual player, we can adjust the rating based on their ratings of similar games from other genres.

3.2 Model Structure

The specifics of the model follow very closely to those outlined in the Global Effects and knn sections of the BigChaos paper [need bib]. Here, we cover the specifics of both.

Note for editing: I know this model is a mess at the moment, just getting the general ideas down on paper.

3.2.1 Global Effects

The motivations for using the formulas that follow can be found in Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights (make this a bib reference). The general idea is to remove the global mean from all ratings to get a residual for each observation. We'll group these residuals by game, and then find an effect for that game. Subtracting this effect from the residual, we'll take those residuals and repeat the same for each user.

As we proceed $y_{u,i}$ will denote the rating by user u for game i. Then, we have

$$y_{u,i} = \mu_{global} + \mu_i + \mu_u + error$$

However, we'll approach this in steps, as follows:

$$\begin{aligned} y_{u,i} &= \mu_{global} + error_{global} \qquad (1)\\ error_{global} &= \mu_i + error_{game} \qquad (2)\\ error_{game} &= \mu_u + error_{user} \qquad (3) \end{aligned}$$

And we'll use the estimation method outlined in the paper (need ref) for each, as follows below:

$$\mu_{global} = \frac{\sum_{u=1}^{N_u} \sum_{i=1}^{N_{games}}}{1} \qquad (4)$$

3.2.2 Classifier Aggrigation

Independent of the global effects, we need to give each user a score in each classification category. We will use 4 types of categorical variables, which will yield (FIND OUT HOW MANY CLASSIFIER VARIABLES). To do this, we can either use the ratings themselves, or the residuals from the global effects process. Let $c_{u,j}$ be the categorical rating for user u in category j. Additionally, for each game i, there exists a set of classifiers C_i that describe the aspects of the game in things such as mechanics, game family, categories, and subdomains. Then for each user, we take all of the categorical values for each user and normalize them, so they should all have mean 0 and standard deviation of 1 for each user. Then, within each category, the average rating is calculated. Now, if a user hasn't rated anything in this category we can assign them a score of 0 in that category (what should amount to an average rating). These serve dual purposes: Firstly, we can assess what sorts of games a user rates higher than average most of the time, and game they rate lower than average most of the time. This will allow the algorithm to not only recommend specific games that have a high predicted rating, but will also allow for the user to find categories or mechanics that they enjoy as well. The other purpose is that we now have a larger dataset to compare people than we might have had before, as we are guarantedd to have (FIND OUT HOW MANY CLASSIFIER VARIABLES) for every user.

3.2.3 k-nearest-neighbors

After we aggregate on the classifiers, we can then find correlations between all of the users. We will then take the k nearest neighbors, where the distance is given by $\rho_{i,k}$, the correlation between the i and kth users. In this algorithm, k will be chosen by a crossvalidation procedure that will be done on a user-by-user basis, as determining a global value for k would be far too computationally expensive. After the optimal k is found, then for each game the k nearest neighbors' (who have rated the game) rating's are averaged using a weighted mean, with the weights being equal to $\frac{2}{1+\rho_{i,k}}$.

3.2.4 Back to Rating

After the k-nearest-neighbors step, we need to "rewind" from the global effects fitting process. For each game, we'll need to add on the specific user effect, the effect for each game, and the overall game mean. This will then yield a rating for each game for each player!

3.3 Parameter Optimization & Selection

There are three parameters that the model relies on: The two α values that balance out biasedness in the global effects section, and the k in the k-nearest neighbors portion of the algorithm. The α's will be done globally (that is, every individual will use the same values for α_1 and α_2), while the values for k will be done locally (each user will have their own value of k). While the approaches for finding the optimal values for these parameters may be slightly different, the general implementation will be the same. We'll first begin by partitioning a subset of the data into 10 different sections (10 'folds'), and then choosing an appropriate parameter space for the parameter to be optimized. Ffor the alpha's, we'll choose values between 0 and 1000 (double check this), evenly spaced, and for k we'll choose a positive integer. We'll then proceed fold by fold, using the other 9 folds to fit a model, and then using the data in current fold to check how the model works. Using root mean square error, we can then rank each of the possible parameter values in the parameter space, and the value with the lowest rmse will be the 'optimal' value for that parameter.

4 Technical Implementation

** SCREENSHOTS NEEDED **

This algorithm is implemented in R using shiny as a vehhicle for a simple, easy to use user interface. Behind the scenes, the data is stored in a data base with six tables: A player key table, which relates the players information to a unique player userid; A game key table, which relates game information to a unique game id; A ratings table which stores the ratings (a score from 1-10) along with the userid of the person who rated it and the game id which the rating is for; the classifiers table, relates the game id to characteristics about the game that will be useful in the classifier aggrigation and k-nearest-neighbors section; the predicted ratings table, which will store predicted ratigs if ratings have already been fit for a particular user; and the aggregated classifier ratings for each user. This data storage will allow for ease of implementation: we only really need the ratings table and the classifier table to do all the work, and the key tables are really only needed to display the inforation to the user in a format which they will understand.

In an implementation of this, the application will begin by having the user enter the username they've registered on `boardgamegeek.com`. Instead of logging the users ratings directly through the application, we will instead leverage the existing infrastructure of boardgame geek. After checking that its a valid username, the application will scour boardgamegeek to see if the database's information is complete: that is, do we have this user's data in the player key? Have they rated games that aren't in our database? Have they rerated games that are in our database? Have they rated games that aren't in our database? .If so, we also need to bring in other users ratings for any games that we may not have had in our database (since a single rating for a game isn't very useful and will greatly skew the results for that game). After this initial step and database update, we can see if they already have saved predicted ratings in the system, and if not, we'll proceed to model fitting.

We first apply the global effects strategy as descibed in the modling section. The optimization of the α parameters will have occred sometime before this and the algoritm will use use those stored values. A weeky or monthly automatic updaing procedure is used to keep these values up to date as more data comes into the system. From the global effects, we then aggregate those based on the classifier aggrigation procedure as described above. These are then stored in the database for future use by the system. After that, before we move onto the k-nearest neighbors implementation, we must first find an optimal value for k using the 10-fold cross validation as described above.

Finally, after the k-fold crossvalidation, we're ready to run the other games through the algorithm to get a predicted score. The games that the user has already rated will then be replaced with those ratings, and the full list will be outputted for the user.

Since we saved the aggregated classifier ratings for the individual user, we can suggest ways for the user to subset this data. By providing the graph below to the player (GET THE GRAPH WORKING), they can see for themselves their not-necessarily obvious preferences in game family or mechanic. They can then search by their preferred mechanic or theme, and get ratings for games that fit that description. Or the user can simply sort by highest rated.

5 Conclusion