Board Game Recommendation

Sam Helmich

March 26, 2015

Abstract

You'll need to include an abstract here

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.

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.

Refer to table 2 somewhere in the text, add a label to the ratings table as well.

GAME_ID	CLASSIFIER	VALUE
1	yearpublished	1986
1	playingtime	240
1	boardgamemechanic	Area Control / Area Influence
1	boardgamemechanic	Auction/Bidding
1	boardgamefamily	Country: Germany
1	boardgamecategory	Dice
1	boardgamemechanic	Dice Rolling
1	boardgamecategory	Economic
1	boardgamemechanic	Hand Management
1	boardgamecategory	Negotiation

Table 2: Example lines of classifiers data, as seen in Database

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.

To keep the information as compact as possible, individual players and games are referred to by an ID number, which corresponds to a row in one of the key tables. These key tables contain information about the players and game that we do not necessarily need to include with each rating or classifier. This greatly reduces the size of the database needed to store and work with the information.

Not all of the data was scraped before creating the applicaiton. Data for 6,088 games was scraped and stored across the four tables. The reason for not scraping all of the data is twofold. First of all, for those 6,088 games there are over 3 million unique ratings. If we were to expand it to all 75,756 games known to BoardGameGeek, then our data set would be crippingly large for all but very high-powered machines. Instead, we implement a continuous data scraping procedure and update the recommender model dynamically: Each time a person uses the app, we make sure to check that all of their current information is correct. We look through known ratings and scrape their userpage on BoardGameGeek. Based on this, we revise old ratings and add new ones, as well as check to see that all of the games that this user has rated are included in the database!

	DATA_USERID	GAME_ID	RATING
1	734	1	10.00
2	820	1	10.00
3	1388	1	10.00
4	332	1	10.00
5	372	1	10.00
6	878	1	10.00
7	2587	1	10.00
8	628	1	10.00
9	5379	1	10.00
10	4899	1	10.00

Table 3: Ratings Data, as seen in Database

	data.userid	data.username	name
1	3	cwmassey	Craig Massey
2	4	greg	Greg Aleknevicus
3	5	gamemark	Mark Jackson
4	8	PBrennan	Patrick Brennan
5	9	JustDebbie	Deborah Pickett
6	17	gschloesser	Greg Schloesser
7	18	njsauer	Nick Sauer
8	21	mbialeck	Mike Bialecki
9	25	jaylorch	Jay Lorch
10	26	Chris Sjoholm	Chris Sjoholm

Table 4: The Player Key relates important info about the individual to the userid

3 Model

3.1 Model Selection

We base our implementation of a recommender system on two papers of on one the most famous recommender systems of all: Netflix. In 2006, Netflix offered a $1 Million prize for an 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, these teams ended up taking something of a shotgun approach, using a linear blend of many different models to come up with a single super-model.

XXX What is the approach you are taking compared to Netflix? the same? similar? what is different? - One or two sentences here.

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 a similar manner to the way that BigChaos implemented the Global Effects model (**?**). XXX the next clause needs an end to be a sentence :) While they implemented 14 steps including Movie Effect, User Effect, and those variables crossed with variables like time since rating, average rating for the movie, production year, standard deviations, as well as others. XXX explain RMSE; make sure in the next paragraph to always point out what is yours and what is netflix. You know all those distinctions, but for any other reader it is not that obvious. From the RMSE of their probe set,

	game_id	name	year.pub	min.players	max.players	playing.time
1	1	Die Macher	1986	3	5	240
2	2	Dragonmaster	1981	3	4	30
3	3	Samurai	1998	2	4	45
4	4	Tal der Knige	1992	2	4	60
5	5	Acquire	1964	3	6	90
6	6	Mare Mediterraneum	1989	2	6	240
7	7	Cathedral	1978	2	2	20
8	8	Lords of Creation	1993	2	5	120
9	9	El Caballero	1998	2	4	90
10	10	Elfenland	1998	2	6	60

Table 5: The Game Key gives useful information about the game

the difference between using only user/movie effcts and using all 14 effects was only .02, a decrease of about 2%. Noting this negligible increase in accuracy and recognizing my lack of significant computing resources, we will only use user and game effects. XXX What is your computing power? How much further could you go with (how much?) more? – How much more accuracy would you hope for with more power?

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 does not 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 Sequence

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.

XXX Describe the sequential nature of the models in the next subsections here.

3.2.1 Global Effects

The general idea behind global effects is to account for the natural biases present in particular games or players. For example, we might have two players who both play and enjoy the same game the same amount. However, one person might give this game a score of 10 while the other gives it a score of 9. Despite their overall enjoyment being the same, perhaps the latter player just tends to be a harsher rater in general. The global effect for this person would then be lower than that of the much more cheery fellow who gave the game a 10. Similarly, this effect can be present for games due to any number of reasons.

We'll estimate these effects one at a time using a heirarchical structure. Define $r^{(1)}_{u,i}$ be the rating from user u for game i. The first effect we'll fit is a global mean, which we'll denote with the parameter θ_g, and we'll model the ratings using the following formula:

$$r^{(1)}_{u,i} = \theta_g + \epsilon^{(1)}_{u,i} \tag{1}$$

θ_g is estimated using a simple mean of the $r^{(1)}_{u,i}$'s. After removing a global mean, we'll model the global effect for each game using the residuals from the global mean step. For this purpose, define $r^{(2)}_{u,i} = \epsilon^{(1)}_{u,i}$, which we'll model using global effect for each game, denoted θ_i. The residuals are then modeled using the following, where x_i is an indicator varialbe that the residual belongs to game i:

$$r^{(2)}_{u,i} = \theta_i x_i + \epsilon^{(2)}_{u,i} \tag{2}$$

To estimate θ_i, we want something that is like a mean, but also contains an offset to avoid overfitting. Following from Bell and Koren (2007), $\widehat{\theta}_i$ is estimated using the following, where α_{game} is a tuning parameter, and the sum is over all residuals for game i and n_i is the number of residuals for game i:

$$\widehat{\theta}_i = \frac{\sum r^{(2)}_{u,i}}{n_i + \alpha_{game}} \tag{3}$$

The final step in the global effects stage is finding the global effect for each user denoted θ_u. In a simillar manner to the game global effects, once again taking the residual from the previous step and model them. To proceed, let $r^{(3)}_{u,i} = \epsilon^{(2)}_{u,i}$, and those residuals are modeled using:

$$r^{(3)}_{u,i} = \theta_u x_u + \epsilon^{(3)}_{u,i} \tag{4}$$

And again, θ_u is estimated with $\widehat{\theta}_u$, which is calculated with:

$$\widehat{\theta}_u = \frac{\sum r^{(2)}_{u,i}}{n_i + \alpha_{player}} \tag{5}$$

Where α_{player} is a tuning parameter to prevent overfitting.

Noting that this can produce esimates that are above 10 and below 1, ratings that fall outside of $[1, 10]$ are truncated to the closest interval endpoint.

3.2.2 Classifier Aggregation

After finding the global effects for the games and the users, we're left with residuals for every rating that every user has made. These residuals represent the last of what differentiate the users based on their preferences for the nuances of each game. We need a way to quantify these nuances, and then attempt to model these residual so we can get a better understanding of each individual user. Each game in the dataset can be described by things like what mechanics it has, or what family of game it falls under, when it was published, etc. All in all, there are up to 1,851 different quantifiable ways that were chosed to descibe games. For example, the game Dragonmaster was published in 1981, has a playingtime of 30 minutes, is under the family "Animals: Dragons", falls under the categories "Card Game", "Fantasy Game", is in the subdomain "Strategy Games", uses the mechanic "Trick-taking", and seats 3 or 4 players. Each one of these will be treated as a classifier for the game.

In order to model the remaining residuals, we'll give each player a score based on how they rated games with similar classifiers; that is, how did they rate games with the "Trick-taking" mechanic, or how did they rate "Card Games"?

Define $C_{u,z}$ to the be the classifier score for user u for classifier z where z is an index of the 1,851 classifiers. Then:

$$C_{u,z} = \frac{\sum \epsilon^{(3)}_{u,i \in z}}{n_z} \tag{6}$$

where the sum is over all residuals that belong to a game that can be described by classifier z. For example, if the classifier was the mechanic "Trick-taking" and the user was $u = 3$, the classifier score would be the mean of all of the $\epsilon^{(3)}_{3,i}$ for *all* games that the user has rated that have the "Trick-taking" mechanic. If its the case that a user has rated no games with a particular classifier, we'll leave it as an unobserved value.

3.2.3 k-nearest-neighbors

Now we have a few classifier values defined for each user who has rated at least one game. In order to create predictions for other games that the user hasn't played, we'll find the correlation between players based on their classifier values. For each pair of players, the correlation between them can be found by finding all of the classifiers that have scores for both players, and then finding the correlation between those values. To make sure that we do not find high correlations between players who only have very few classifier scores in common, we'll on consider a correlation between players if they have more than 6 common classifier scores.

Now to generate an estimate for user u for game i, we'll find all of the players who have a correlation value with user u who have also rated game i. The rating for game i is then the weighted average of the k most highly correlated (or nearest) neighbors of user u.

The weights for this will be $\frac{1}{1+\rho_{u,k}}$ where $\rho_{u,k}$ is the correlation between user u and user k. The number of neighbors to use k will be determined by cross-validation on a user-to-user basis. In the end $z_{u,i}$ will be the k-nearest-neighbors estimate for user u and game i, as given in the equation below (where $u_{[j]}$ is the user with the j^{th} highest correlation between themselves and user u:

$$z_{u,i} = \frac{\sum_{j=1}^{k} \epsilon^{(3)}_{u_{[j]},i} * \frac{1}{1+\rho_{u,u_{[j]}}}}{\sum_{j=1}^{k} \frac{1}{1+\rho_{u,u_{[j]}}}} \tag{7}$$

3.2.4 Back to Rating

From what we've produced above, of the estimated rating for user u and game i is given by:

$$\widehat{r}_{u,i} = \widehat{\theta}_g + \widehat{\theta}_i + \widehat{\theta}_u + z_{u,i} \tag{8}$$

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_{game} and α_{player}), while the values for k will be done locally (each user will have their own value of k).

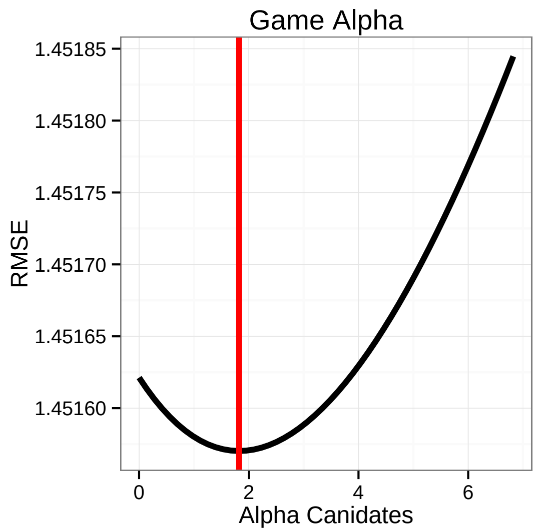

Figure 1: RMSE of candidate α_{game} values. XXX there's a typo in the x label

3.3.1 Optimizing tuning parameters α

XXX avoid using symbols without also calling them by their names ... i.e. say tuning parameter α rather than just α.

To optimize the α values, we follow the model fitting procedure up to the point in which the paramter is used, estimate the model parameter using a grid of possible values for the turning parameter, and then find which value for the tuning parameter yields the model parameter estimate which has the lowest root mean square error (RMSE). XXX back this definition of RMSE up with a formula and move it to the first time where you mention RMSE. RMSE is calculated by finding the difference between the rating and the rating that is estimated using only the global effects parameter(s).

What does 'the first'meane here? Does it mean that there is a given order, which we have to follow, or did you decide to first tune the game parameter? Does it matter which parameter is tuned first? The first α to be tuned is α_{game}. XXX What you are describing here is called ten-fold cross validation. The data is randomly divided into 10 separate sections of roughly equal size. Then for each of the 10 sections, we'll treat the section as a test set and the other 9 to use as a training set. $\widehat{\theta}_g$ is fit first, and then from a grid of α_{game} values (where $\alpha_{game} \in [1, 10000]$), a value of $\widehat{\theta}_i$ is produced for each of the α_{game} in the grid. For each of the $\widehat{\theta}_i$ that are estimated, we'll use the section that wasn't in the training set, the RMSE is calculated by:

$$\text{RMSE} = \frac{\sum r_{u,i}^{(1)} - (\theta_g + \theta_i)}{n} \tag{9}$$

where we sum over all rated games in the test set, and n is the number of ratings in the test set. The value of α_{game} that has the lowest RMSE will be used. in figure 1, we see the optimal value for α_{game} sits in a nice trough, with the minimum at $\alpha_{game} = 1.802303$. XXX do you really need that many significant digits?

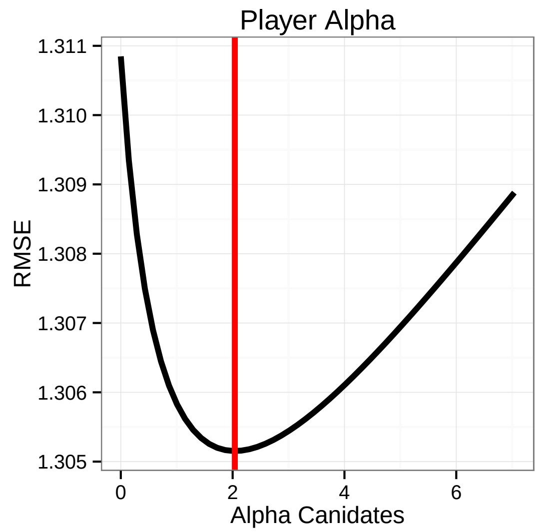

Figure 2: RMSE of candidate α_{player} values. typo in X

After finding the optimal value for α_{game} we can move on to optimizing α_{player}. Once again, the data will be divided randomly into approximately 10 equally sized sections, and each will be treated as an individual test set while the other 9 combined will serve as a training set. A grid of $\alpha_{player} \in [1, 10000]$ will be used to create a value for $\widehat{\theta}_u$, and for each of those values, we'll calculate the RMSE as below:

$$\text{RMSE} = \frac{\sum r_{u,i}^{(1)} - (\theta_g + \theta_i + \theta_u)}{n} \tag{10}$$

Once again, the value of α_{player} with the lowest RMSE will be the value used, which as can be seen in 2 is $\alpha_{player} = 2.022021$.

One may immediately notice that different α candidates have a larger effect on RMSE for players than they have for games. The reason for this is that there are far more ratings for each game than there are for each player. This results in less overfitting for each game and more overfitting for each player, as is seen in the histograms for the number of ratings for each game and player.

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 aggregation 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

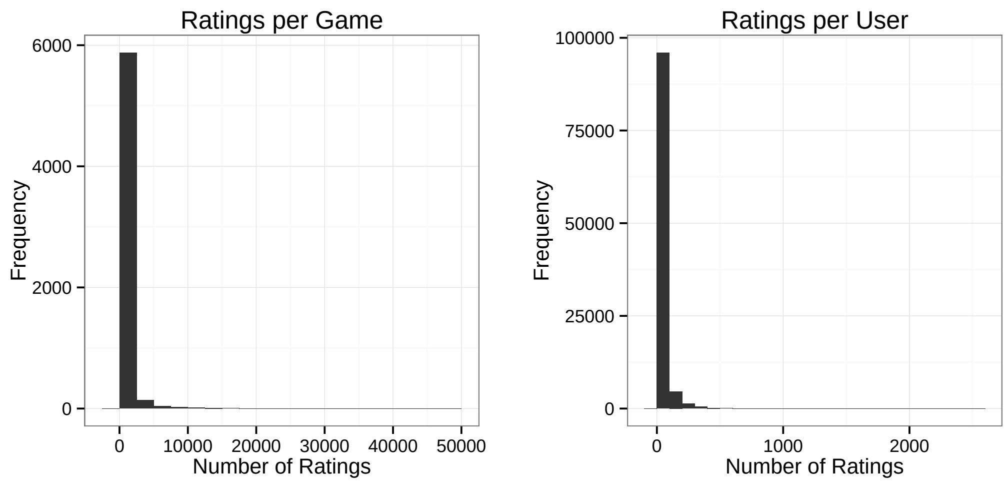

Figure 3: Frequency of number of ratings for games (left) amd players (right). A log transformation on the frequencies might be more informative. Make sure to emphasize it, if you show transformed data.

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 aggregation 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.

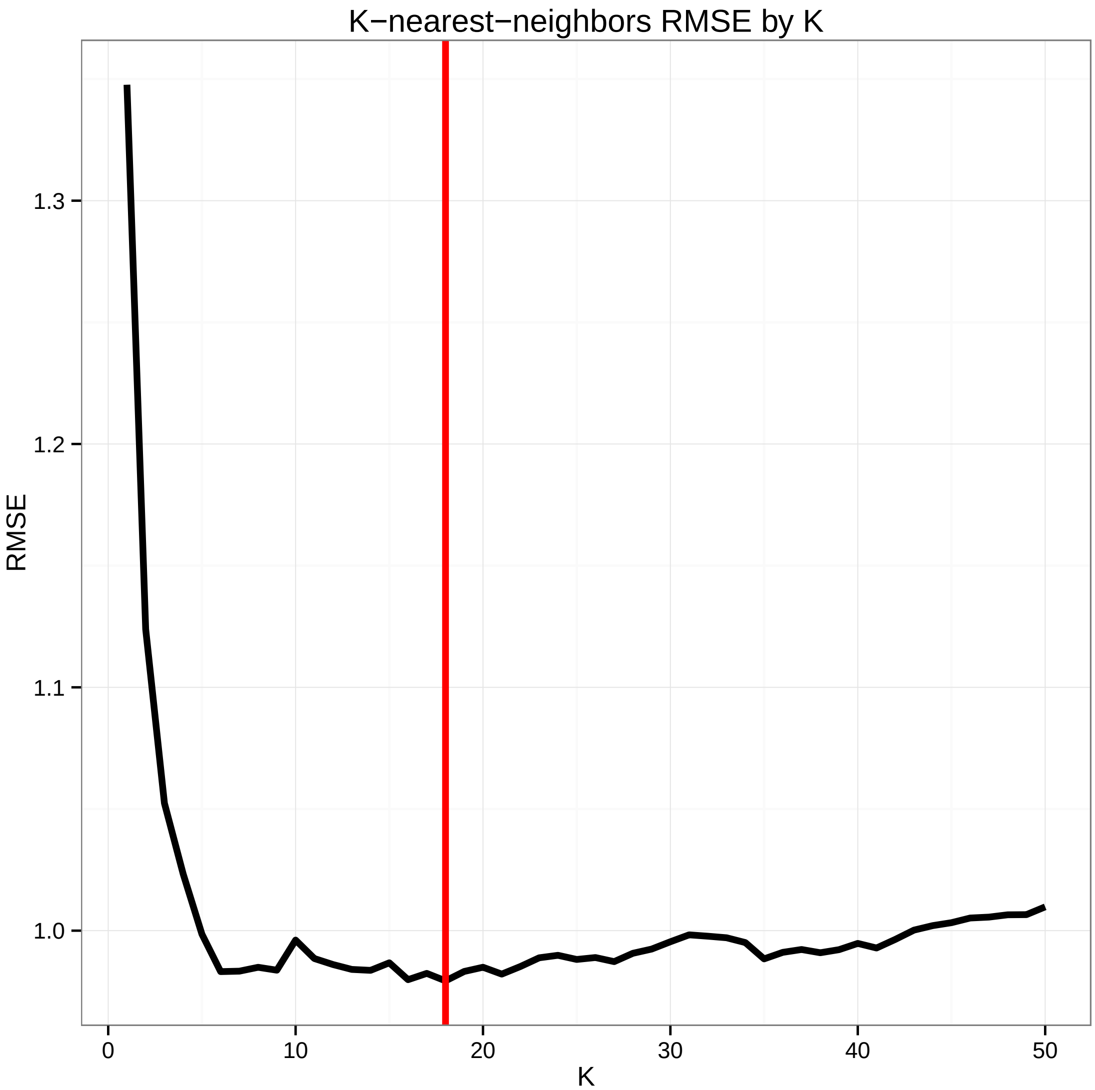

Figure 4:

5 Conclusion

References

Bell, R. M. and Koren, Y. (2007), "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights," .