Ranking Products by Bayesian Estimates

JONATHAN BHASKAR AND PRATIK GOSWAMI

Problem:

A lot of Web applications today have user ratings. A few examples would be the product reviews on Amazon, the restaurant ratings on Yelp, movie ratings on IMDB, or the Hotel ratings on Trip Advisor. These kinds of web applications also commonly rank their products based on these ratings. There are two common types of rankings:

1. **Popularity:** This ranking is usually based on the number of reviews, the number of times the particular product is visited or sold among other criteria. This ranking is not usually based on the average rating of the product itself. So it's very common to see a very low rated product with a lot of ratings ranked higher than a product with fewer but much better ratings. (E.g. Most Reviewed on Yelp, Popularity on Amazon)

2. **Rating:** This ranking is based on the average rating of the products. Again for this ranking, the web applications do not consider the number of votes. So a product that has a 5/5 rating with five reviews would be rated higher than a product with an average rating of 4.5/5 and five hundred reviews. (E.g. Average rating on Amazon, Highest Rated on Yelp).

In this project we compare some of the ways to consolidate both these criteria into one single ranking. The challenges in doing this are:

1. There is no easy way to do this. The most popular way today is using the Bayesian average, which requires an understanding of Bayesian probability, a non-trivial and unintuitive system.
2. The quality of a product is subjective and hence, so is the ranking. Therefore guessing what type of ranking system works best is a hard task.

Methodology:

Bayesian Average:

We started off with the MovieLens database. We tried the Bayesian Average, which is currently used by IMDB and Rotten Tomatoes. The formula is:

$$X = \frac{C \times m + n \times r}{C + n}$$

Where:

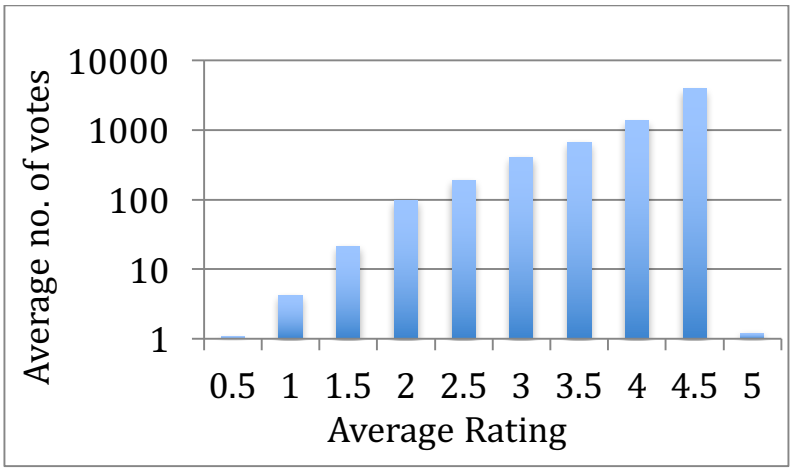
X is the Bayesian average of the product
C is the prior number of votes
m is the prior rating (average)
n is the number of votes for the product
r is the average rating for the product

We used the average across all products as m and found it to be 3.13/5. C is the minimum number of votes us to be confident about a product's rating i.e. the product's rating has converged to its average and is not going to change a lot in the future.

One possible value for C could be the average number of ratings across all products. We found this value to be close to 800 for this dataset. But could we have confidence on the rating with even fewer votes? To find this, we plotted the average number of votes against the average rating of products (Graph on next page).

We also found that a product needed only around 400 votes to reach the average of 3.13. So 400 could also be a potentially good value for C.

Also intuitively, we found that the extreme ratings (0.5 and 5) had a low number of reviews. Movies with good ratings (4.5) are more popular than the rest.



Dirichlet Prior:

One major problem with the Bayesian Average is that it assumes an underlying normal distribution and that every rating has a value on that distribution. This might not be true. For example, we could be confident that a movie with 100 votes and an average rating of 1/5 is bad. But with the Bayesian average and C=400, the result would still be the average of 400 votes of the average rating (3.13) and the 100 votes of the product (which is equal to 2.7). If a rating does not have enough votes, it is pushed towards the mean no matter what its ratings are.

A multinomial distribution with dirichlet priors for the number of votes of each star (number of one star ratings, number of two star ratings, etc.) solves this problem. The equation is:

$$X = \sum_{k=1}^{K} k \frac{n_k + 1}{N + K}$$

Where:

X is the average
K is the number of possible ratings
n_k is the number of ratings for each k
N is the total ratings for the product

Results:

The top 20 movies sorted by their dirichlet rank are as follows:

Name	Ave. Rating	No. of ratings	Bayesian C=400	Bayesian C=800	Dirichlet
Shawshank Redemption	4.448	67741	1	1	1
The Godfather	4.361	44411	2	2	2
The Usual Suspects	4.334	48239	3	3	3
Schindler's List	4.313	53609	4	4	4
Seven Samurai	4.273	11796	9	11	5
The Godfather: Part II	4.272	29198	5	5	6
Rear Window	4.264	17996	6	8	7
Casablanca	4.254	26114	8	7	8
One Flew Over the Cuckoo's Nest	4.252	32948	7	6	9
Sunset Boulevard	4.242	6903	17	33	10
Dr. Strangelove	4.24	23915	10	9	11
The Third Man	4.236	6861	21	34	12
City of God	4.234	13142	15	17	13
The Lives of Others	4.326	5825	27	42	14
North by Northwest	4.229	16976	13	14	15
Paths of Glory	4.229	3598	46	75	16
The Dark Knight	4.225	21192	12	13	17
Fight Club	4.223	41409	11	10	18
Double Indemnity	4.225	4988	36	50	19
12 Angry Men	4.222	13235	16	21	20

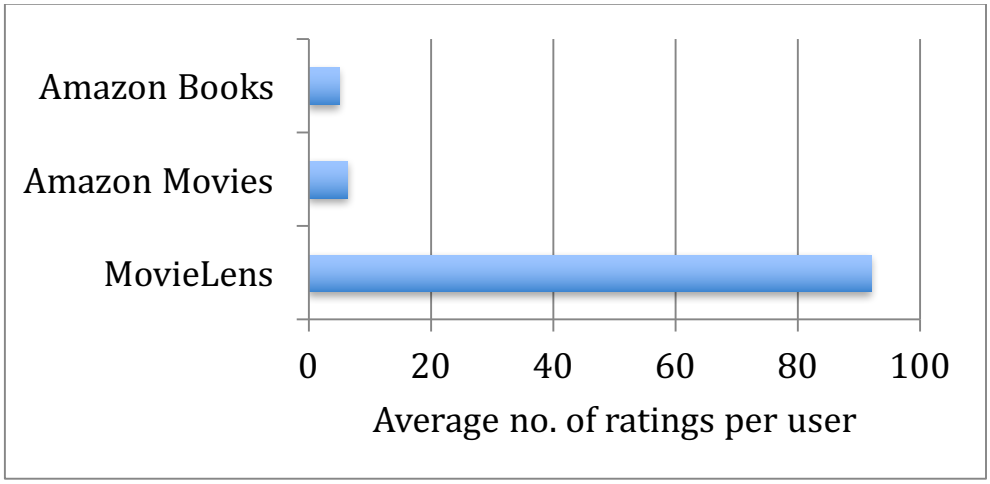
Almost all the movies in our top 20 are widely considered to be some of the best movies of all time, and all of them are in the IMDB top 250. So both the algorithms do a pretty good job at the ranking.

The key difference here is the number of votes a movie needs to make it to the top of the rankings i.e. the weight the ranking system gives to the number of votes. It looks like the dirichlet priors give the lowest weight to the number of votes, followed by the Bayesian average with C=400 and C=800. But a point to note is that the dirichlet priors do not blindly consider the number of votes, but rather how the votes are distributed (number of one star votes, number of two star votes, etc.)

Amazon Data:

After working with the MovieLens data, we tried the same algorithms with the Amazon Books and Movies data. But to our surprise we found that the rankings did not work well for these datasets. The top rankings were not very popular or good books/movies.

So we tried to figure out why this was happening. Our best explanation for this is because of the average number of reviews per user.



On average, a user has rated over 90 movies in the MovieLens dataset, whereas the Amazon books and movies datasets have only 3-4 ratings per user. As we're dealing with rankings, there has to be some common ground between products for a valid comparison. This common ground comes from a lot of common users rating the same two products. If a user only rate few products, their tastes are not captured by the dataset, and in turn, products are independent of each other and cannot be compared.

Complexity:

We used Spark for our analysis. For this project, spark goes over the entire dataset a constant number of times, and does a constant amount of work on each line of the dataset each time, the time complexity is of an order of n. Spark stores the dataset in memory if need be, and so the space complexity too is of an order of n.

Conclusion:

While the theory behind these rating systems are not very easy to understand, they do a great job at ranking items given certain constraints (which I'm sure can be overcome with more analysis). Even today, only a few web applications have proper ranking systems, and there is room for improvement. Showing customers the best products can improve sales and interest (clicks). Implementing these systems would be a big step forward in the right direction.

References and Datasets:

- Movielens: http://grouplens.org/datasets/movielens/
- Amazon: http://snap.stanford.edu/data/web-Amazon-links.html
- http://www.evanmiller.org/ranking-items-with-star-ratings.html
- http://stackoverflow.com/questions/1411199/what-is-a-better-way-to-sort-by-a-5-star-rating
- https://districtdatalabs.silvrback.com/computing-a-bayesian-estimate-of-star-rating-means

Code:

- amazon_process_file: Filter only needed fields from input file
- amazon.py and movielens.py: Contains a majority of the computations
- amazon_user_average and movielens_user_average: Computes the average number of votes per user