Book Recommendation System

Jinny Cho Ryan Gorey Sofia Serrano Shatian Wang
JordiKai Watanabe-Inouye

Advisor: Prof. Anna Rafferty

Winter 2016

1 Introduction

Recommendation systems are used in hundreds of different services - everywhere from online shopping to music to movies. For instance, the online retailer Amazon had a heavy hand in developing collaborative filtering algorithms that recommend items to users. Music services like Pandora identify up to 450 uniquely identifying characteristics of songs to find music similar to that of their users' preferences. Other music streaming services, such as Spotify, heavily rely upon the music selections of similar users to make weekly song recommendations and personalized radio stations. Netflix, a popular television and movie streaming service, uses these systems to recommend movies that viewers may enjoy. We can see how recommendation systems have a surprisingly large impact on the materials consumers engage with over the course of their daily lives.

For our senior comprehensive project, we analyzed three systems that predict how users will rate specific books. Our system that we created makes these predictions based on data gathered from the Amazon Book Reviews dataset, BookCrossing dataset, GoogleBooks API, and GoodReads API. To accurately predict users' reactions to books, we've integrated several strategies in the field of recommendation systems.

2 Background

There are three major approaches for recommendation systems: (i) content-based, (ii) collaborative, and (iii) hybrid. Broadly, recommendation systems that implement a content-based (CB) approach recommend items to a user that are similar to the ones the user preferred in the past. On the other hand, recommendation systems that implement collaborative filtering (CF) predict users' preferences by analyzing relationships between users and interdependencies among items; from these, they extrapolate new associations. Finally, hybrid approaches meld content-based and collaborative approaches, which have complementary strengths and weaknesses, thus producing stronger results.

3 Data

For this project, we used two datasets: BookCrossing (BX) and Amazon Book Reviews (AB). Users don't tend to rate or read many books, so one can imagine how inherently sparse our data was. In order to alleviate this sparsity, we used the intersection (by books) of these two datasets for our project. The intention was to increase the number of ratings each book had. The intersection of both datasets resulted in 36,493 books. There are 283,480 unique users in the Amazon dataset that we used and 951,578 ratings and reviews; there are 37,830 unique users in the BX dataset that we used, along with 258,585 ratings. That gave us a total of 321,310 users in the intersection.

However, each algorithm has constraints that caused us to filter out any users who had rated less than three books. This left us with 35,126 books and 98,261 users. Then, we partitioned our data into training, development, and evaluation sets. We considered each user and put 60% of their ratings in training, 20% in

development, and 20% into evaluation. This allowed us to tune different variables, finding the most accurate combination without overfitting, which partitioning as described helps to prevent.

The BX dataset contains 271,379 book ratings. On its own, this dataset would have limited us to a CF approach. The BX dataset does not contain any fields relating to content (i.e. review text or book description). So we supplemented this dataset with data from Amazon. The AB dataset includes not only user ratings but also at least five reviews (per user). By providing us with text we could use to model a book, these reviews allowed us to explore a CB approach. In addition, we retrieved book descriptions from two APIs: Google Books & GoodReads. These descriptions, along with the Amazon text reviews, were used in the CB approach. In addition, we used the ENABLE dictionary of English words as our source of words for text-based modeling of the books.

4 Content-Based Approach

Overview

CB recommendation strategies consist of a few high-level steps: First, a quantitative model is generated to represent each of the products. The model should represent something about the actual content of the product. Next, using the model and feedback (usually in the form of user ratings or reviews), the algorithm builds a user profile. The profile is essentially a classifier that learns how to accurately predict a user's ratings for the modeled items. Finally, each of the modeled products a user has not yet tried is presented to the user profile to determine an approximate rating, and the highest-rated objects become the most highly recommended items for that user.

Because CB algorithms do not necessarily require user ratings of a product in order to build their representation of that product, CB recommendation is particularly useful on datasets where many products have relatively few ratings or no ratings. We later refer to this as 'sparsity'. However, since it only recommends products to a user based on the model generated from what they have already rated, CB recommendation struggles with what is termed the "serendipity problem": users are unlikely to be recommended new items that are dissimilar to items they have already rated, and that they would therefore not have found for themselves. [3]

Methods

Our first task was to model the books in our datasets. We chose two different approaches to doing so, both of which produced one vector of real numbers per book. For both approaches, we preprocessed the text of all descriptions and reviews we had for a particular book to remove punctuation, ensure all letters were lowercase, and lemmatize each word to its base form.

The first was a term frequency-inverse document frequency (TFIDF) representation of each book, which scored the importance to the book of each of the 33,986 words in our lemmatized version of the ENABLE dictionary without stop words, while accounting for how discriminatory that given term would be across all of the books. [9] We also tried a more experimental approach that went beyond what we found in the literature, applying aspect sentiment analysis to tens of thousands of nouns from our lemmatized dictionary, and averaging a score for each word appearing in the text describing the book we were modeling. We decided to try this approach because the vast majority of the text we had describing any given book consisted of very opinionated reviews, and trying a method of modeling the books that leveraged that fact made sense. In order to do so, we manually went through a list of the most common stop words and phrases, setting aside those that functioned as modifiers for the word directly following them (i.e., "very" or "not") and those across which sentence sentiment tends to be negated (such as "however" or "but", yet leaving out phrases such as "but also"). Those were the modifiers and sentence-level hinge words that we used in applying the traditional aspect sentiment analysis algorithm, [2] in conjunction with our collection of opinion words, to our text.

Our second task was to use the modeled books to learn a user profile for any user using classifiers. A decision we made early on after reading Mooney and Roy's work on content-based recommendation systems was to train our classifiers to recognize only two classes, books that a user would like and books that they would dislike [4]. This choice also helped to address our very sparse training data for any particular user. We then tested two different types of classifiers in our project: a Naive Bayes classifier and a Maximum

Entropy classifier. Within either class, our implementation of a Naive Bayes classifier estimated a normal distribution based on the sample of features in that class that it received from a user. As for our Maximum Entropy model, we used the SKLearn package's implementation of logistic regression, which uses the very fast liblinear library to train feature weights.

The final piece of the content-based system that we implemented was feature selection based on the set of books that a user had rated in our dataset. Though we could have used every feature in whichever vector(s) our classifiers were considering, there were so many features in any given book vector and so little training data that it made more sense to apply the heuristic of choosing the top 100 features that, across the books a user has read in our training dataset, differed the most on average between their well-rated and poorly-rated samples. In addition, when putting together the features to pass into our maximum entropy classifier, we converted each feature to binary by determining whether it fell above or below the average value of that feature given the class in our training data.

We ran tests to determine which combination of book models and classifiers performed the best. The two classifiers we tested, as stated, were the Naive Bayes classifier and the Maximum Entropy classifier, but we tested four different variations on modeling the books. Two of those variations were the plain TFIDF vectors and the aspect sentiment analysis vectors, but we also wanted to try combining elements of both as features. To this end, we tried two different approaches to combining elements of both: the first was to select the top 50 features (via our previously stated heuristic) from each, and the second was to first identify the top TFIDF features, and then include them and their sentiment-analysis counterparts for the same word until we had accumulated 100 features.

Results

Because we used classifiers to make predictions about whether or not users will like or dislike books, we chose to evaluate our results with three statistics: accuracy, precision, and recall. The accuracy of our classifier tells us the proportion of predictions we make that are correct, including correctly guessing a user would like a book, and correctly guessing a user would dislike a book:

$$\frac{\text{True Positives} + \text{True Negatives}}{\text{All Predictions}}$$

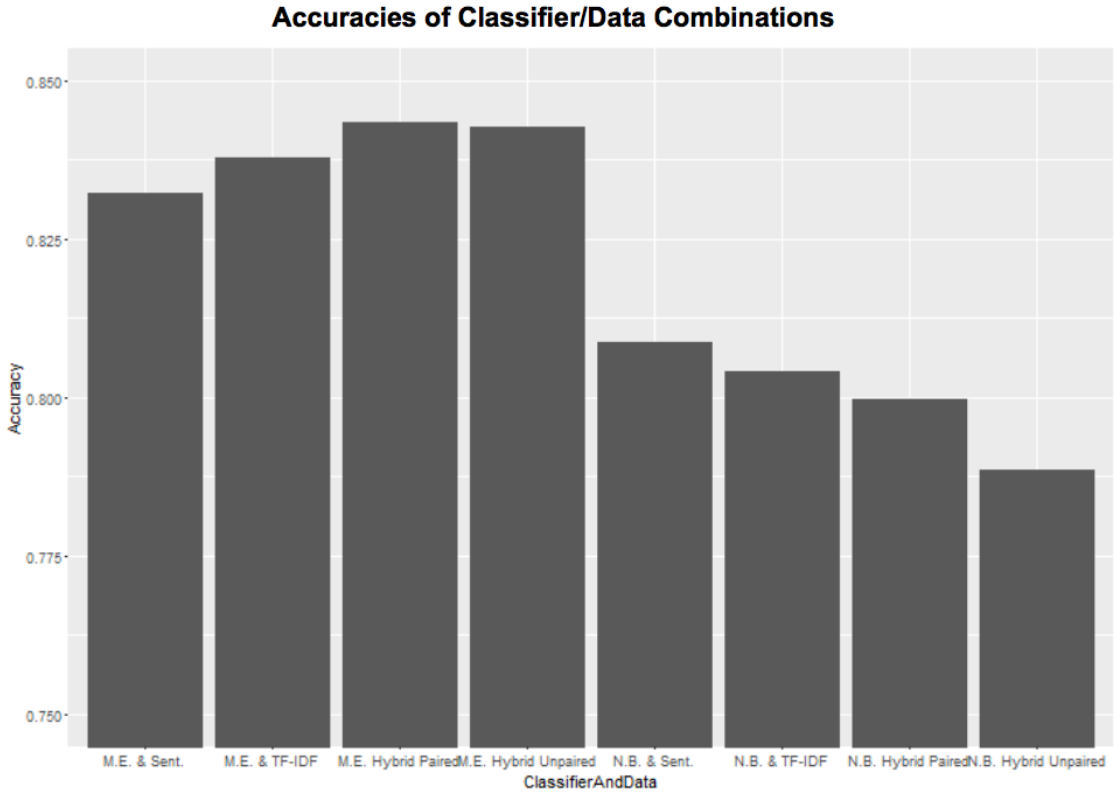
The precision tells us how many books were actually liked by a user out of all the books we predict that user would like:

$$\frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

The recall tells us how many books our classifier predicts users would like out of all the books those users would actually like:

$$\frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

These three statistics can be interpreted together to give us a thorough understanding of the strengths and weaknesses of our content-based rating prediction system. When choosing between classifiers and data combinations, we chose to prioritize the accuracies. Below, we see how the eight combinations of data types and classifiers we chose compare:



We see that most powerful combination of classifiers and feature vectors are the maximum entropy classifier learning using the TF-IDF selected features and associated sentiment scores. In our development data, this combination has an accuracy of 84.35%, meaning that it correctly guesses whether a user likes or dislikes a book 84.35% of the time. It has a precision of 88.77%, meaning that a user would like actually like the books this system predicts they would in almost nine out of ten recommendations. Lastly, its recall is 91.09%, meaning we're correctly capturing 91.09% of the books the user would actually like.

Once we identify the most powerful combination, we evaluate our evaluation data with the same classifier and feature selection method at both 100 features, and 200 features. We found that our performance did not decline when we left the development set.

Table 1: Evaluation Data Results

Number of Features	Accuracy	Precision	Recal
100	84.34%	88.66%	91.16%

The respective statistics remain about constant relative to our performance on the training data. We ended with an accuracy of 84.3%, a precision of 88.66%, and a recall of 91.16%.

5 Collaborative Filtering Approach

Overview

While CB systems recommend items with similar features to users (e.g. books with similar contents or writing styles), CF systems predict user preferences by analyzing past relationships between users and interdependencies among items [8]. More specifically, CF works under the belief that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person. CF systems can be categorized into either non-probabilistic or probabilistic

algorithms. There are a large number of CF algorithms. We've explored two: UV-decomposition and k nearest neighbor.

Methods

UV-Decomposition

As a recommendation system, we'd like to know what the values of the missing entries would be if the user were to actually rate a particular book. However, because users don't tend to read many books, the user-book matrix is approximately 98% sparse. To alleviate this issue of sparsity, UV-decomposition is an algorithm that uses dimensionality reduction, which finds a way to represent the matrix using a much smaller number of values than the total number of entries. Broadly dimensionality reduction would "remove unrepresentative or insignificant users or items to reduce the dimensions of the user-item matrix directly" [8]. UVD is an instance of a more general theory called SVD (singular-value decomposition).

Consider the utility matrix M below, with m rows and n columns. Dimensionality reduction tells us that there exists some decomposition of M into two matrices U and V with dimensions $m \times d$ and $d \times n$. The product UV closely approximates M.

$$\mathrm{M} = \begin{array}{c} \\ book_1 \\ book_2 \\ \vdots \\ book_m \end{array} \begin{array}{c} \begin{array}{cccc} user_1 & user_2 & \cdots & user_n \end{array} \\ \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mn} \end{bmatrix} \end{array}$$

For instance, consider a reduction to a lower dimensional space where $d = 2$. U considers all books. Essentially the characteristics of books is captured the with two numbers. Each of these entries represents the extent to which users value a feature in a given book. Similarly, V considers all users. V generalizes the sort of books a user may like or dislike with only two features. Therefore the vectors for each book and each user are capturing patterns across books and individuals.

$$\begin{array}{c} \\ book_1 \\ book_2 \\ \vdots \\ book_m \end{array} \begin{array}{c} \begin{array}{cccc} user_1 & user_2 & \cdots & user_n \end{array} \\ \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mn} \end{bmatrix} \end{array} = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ \vdots & \vdots \\ u_{m1} & u_{m2} \end{bmatrix} \times \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \end{bmatrix}$$

$$\mathrm{M} \qquad\qquad \mathrm{U} \qquad\qquad \mathrm{V}$$

This algorithm incrementally adjusts a single element of U or V, in order to find a decomposition of UV with least root mean square error. The technique used for finding UVD is an example of gradient descent [1].

K Nearest Neighbors (kNN)

The second collaborative filtering algorithm we implemented is a memory-based algorithm known as kNN. Nearest neighbor algorithms are a family of popular non-probabilistic CF techniques. This technique computes the similarity between either users (user-to-user based CFs) or items (item-to-item based CFs) that are represented by columns or rows in a user-item matrix with similarity metrics such as cosine, adjusted cosine, or euclidean. This algorithm then predicts unknown ratings based on known ratings of similar users or items. The similarity values "provide the means to give more or less importance to these neighbors in the prediction"[5].

Algorithmic Details

The k Nearest Neighbors algorithm takes in a user, book pair, (u_i, b_j) as input and outputs a prediction of user u_i's unknown rating to book b_j. For the book rating prediction task, there are two different ways to implement the k Nearest Neighbors algorithm. One is user-based, and the other one is item-based. User-based kNN looks at all the users who have rated book b_j, and find the k users among these co-rated users that are most similar to u_i. These k users are called the k nearest neighbors of u_i, and with the belief that similar users rate books similarly, their ratings to book b_j are used to predict u_i's rating to b_j.

The item-based kNN, on the other hand, first finds the set of all the books that user u_i has rated, and from this set, picks k books that are the most similar to book b_j (the k nearest neighbors in this case are these k most similar books). User u_i's ratings to these k nearest neighbors are then used to predict u_i's rating to book b_j. The intuition behind the item-based kNN is that a user tends to rate similar books similarly.

We decided to use the item-based kNN since the relations between books are more static than the relations between users [6]. Also, there are around 10 times more users than books in our dataset, which made comparing the similarities between books likely take less time then comparing the similarities between users.

More specifically, our implementation of the item-based kNN takes in a list of user, book tuples $[(u'_1, b'_1), (u'_2, b'_2), \ldots, (u'_n, b'_n)]$ as input, and outputs a list of predicted ratings, the ith entry of which corresponds to the predicted rating of u'_i to b'_i in the input list. For each user, book tuple (u'_i, b'_i) in the input list, our implementation first finds the books u'_i has rated. Then for each book b_t already rated by u'_i, we compute the similarity between b'_i and b_t: we grab the two row vectors in the utility matrix that corresponds to b'_i and b_t. We then use either cosine similarity, adjusted cosine similarity, or Euclidean distance similarity metric to compute a similarity score for the two vectors. This similarity score then quantifies the similarity between b'_i and b_t. The formula for computing the similarity score of two vectors using the above-mentioned three similarity metrics are stated below. Recall that we only consider the co-rated entries for each vector. In the formula below, U is the set of co-rated users who have rated both book i and j, $R_{u,i}$ is the known rating of user u to book i, and $\bar{R}_u$ is the average rating of user u.[6]

$$sim_{\text{cosine}}(i,j) = \frac{\sum_{u \in U} R_{u,i} * R_{u,j}}{\sqrt{\sum_{u \in U} R_{u,i}^2}\sqrt{\sum_{u \in U} R_{u,j}^2}}$$

$$sim_{\text{adjusted cosine}}(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_u)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_u)^2}}$$

$$sim_{\text{euclidean}}(i,j) = \frac{1}{\sqrt{\sum_{u \in U}(R_{u,i} - R_{u,j})^2 + 1}}$$

Once we have done this for all the rated books, we pick k rated books that have the highest similarity scores, and call them the k nearest neighbors of b'_i. We then compute a weighted sum of u'_i's ratings to these k books, with weights being proportional to the similarity scores. This weighted sum is the algorithm's predicted rating of u'_i to b'_i:

$$\text{predicted rating } u'_i \to b'_i = \frac{\sum_{j=1}^{k} sim(b^{(j)}, b'_i) \times \text{rating } u_i \to b^{(j)}}{\sum_{j=1}^{k} sim(b^{(j)}, b'_i)},$$

where $b^{(j)}$ is the jth nearest neighbor of book b_i.

We also observed that different users rate books very differently. Some users rated all the books with the highest possible ratings of 5, while some others rated most books with the lowest possible ratings of 1. This makes comparing ratings across different users difficult. We can normalize the ratings by users to account for this potential issue: for each user u_i, we compute his average rating $\hat{r}_i$. We then subtract from every rating of this user this average rating. This normalization trick makes the normalized average rating of each user 0. Note that the adjusted cosine similarity metric is supposed to take into consideration users' different rating behaviors.

Tuning Parameters

Based on our algorithm above, there are three parameters that we need to tune: the number of neighbors k, similarity measurements, and normalization. It follows that we have a myriad of combinations to test. We have $k = 2, 3, \ldots, 30$ neighbors [7], cosine similarity, adjusted cosine similarity (also known as the Pearson correlation), and euclidean distance as a similarity metric, and normalization of the data. When are then left with a total of 174 distinct combinations. We choose the best combination to test the performance of our algorithm with the evaluation dataset.

More specifically, as we mentioned above, given the user ratings, we split them into three parts: training, development and evaluation. Then using training dataset, we use 174 different combinations to predict the ratings of books and users that are in development dataset. Then we compute the RMSE (Root Mean Square Error), which measures of how far the prediction is off from the actual rating. Then among the 174 different RMSE values, we choose the best combination, which contains the lowest RMSE value. We use this combination to test on the evaluation data. The following section explains more detailed numerical results of this operation.

Results

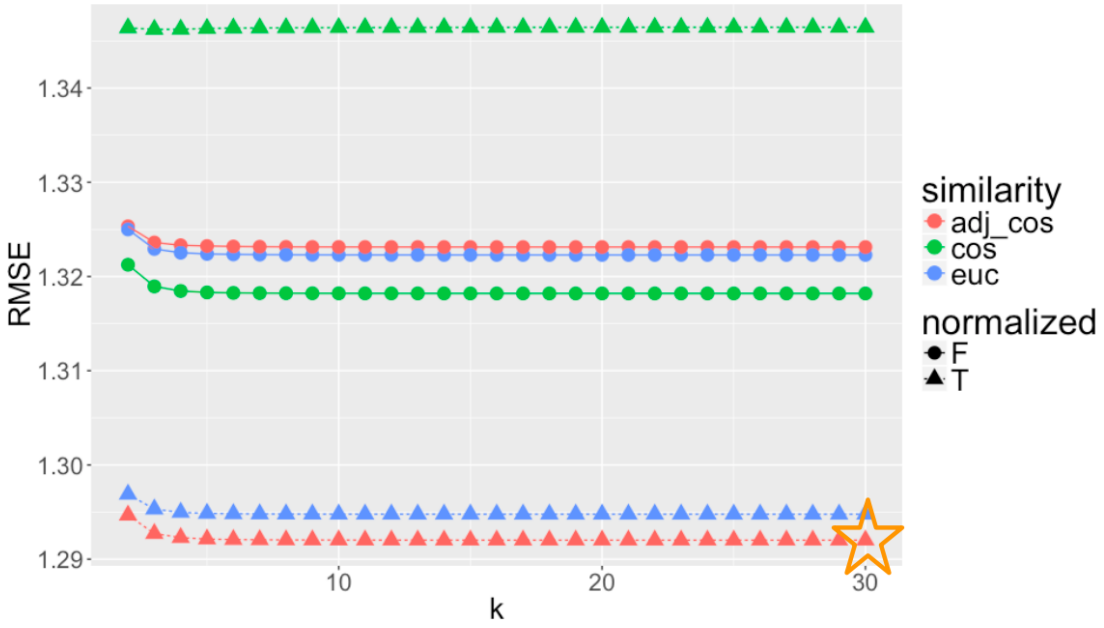
We used the evaluation metric known as the root mean square error. This tells us on average how different our predictions are from the actual rating a user has given.
Evaluation Metric: RMSE

We calculate the root mean squared error using the following equation

$$RMSE = \sqrt{\frac{1}{n}\sum_{(i,j)}(p_{(i,j)} - r_{(i,j)})^2}$$

Where n is the total number of ratings, $p_{(i,j)}$ is the predicted rating and $r_{(i,j)}$ is the actual rating for a given book j, user i pairing. The RMSE value indicates on average the difference between our model's prediction from the actual rating observed. [8].

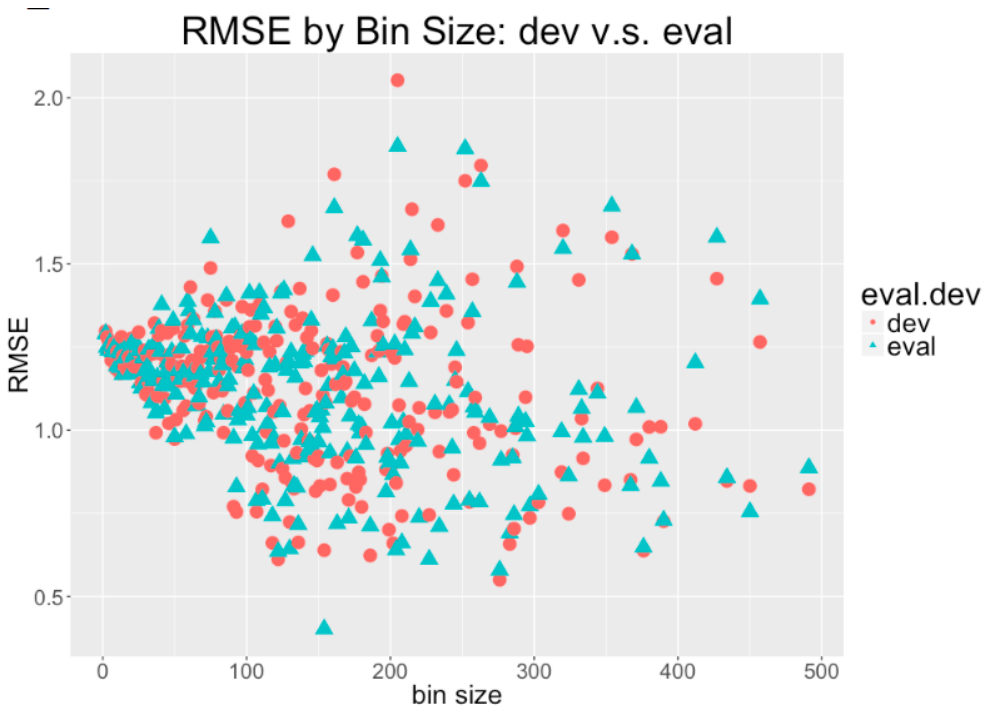
The predicted ratings for the development data are informed by users' preference from the training data. We use this evaluation metric to find the best combination of variables (number of neighbors, similarity metric, and normalization) for predicting ratings.



What we would like to do now is compare this nearest neighbor algorithm to a couple of baselines. One of the baselines we generate was predicting random ratings; the other is predicting ratings based on a user's

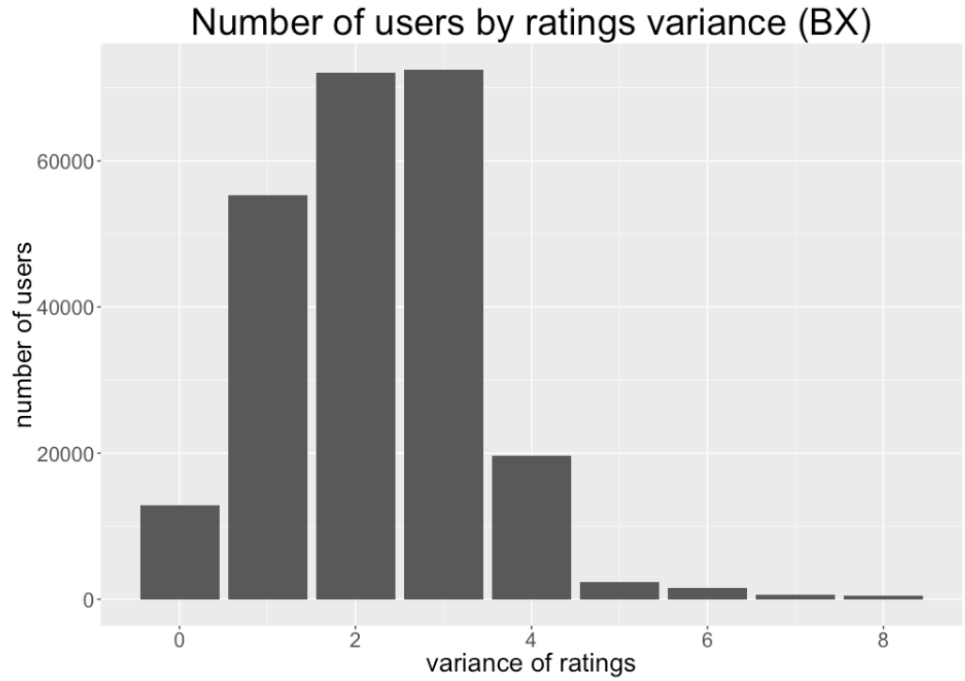
average. We found that although we do better than predicting random ratings, we do not do better than predicting a user's average. However this can be explained by anomalies in the data.

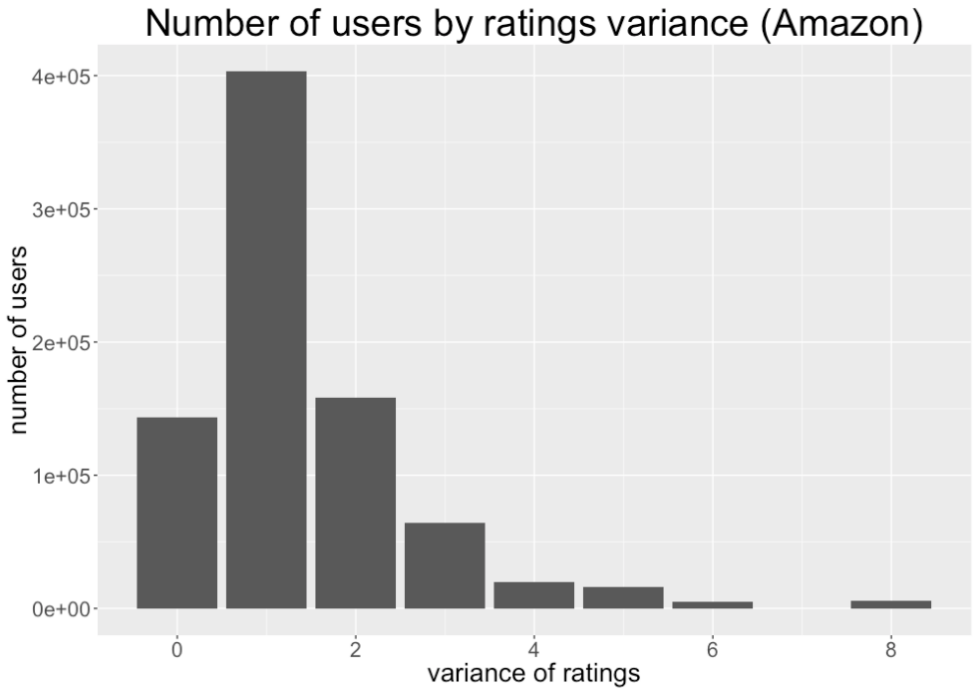
Before explaining the anomalies, using this best combination, we then predict ratings in the evaluation dataset. Below we compare the RMSE values from the development data to the evaluation data. Recall that we tuned our variables with the development data. Specifically, the graph below compares the number of ratings known from the training set to the RMSE value. For instance, there exist some users who've rated 150 books and their RMSE value in the evaluation set is approximately 0.25.



One would expect that the greater the number of books a user has rated the better the prediction for that user are. However, this graph exhibits otherwise. This graph shows that the number of ratings a user has given is irrelevant.

These results are peculiar, if we look at them alone. However, if we look at the nature of our data, it actually makes quite some sense. The variance of ratings by users is particularly low for both datasets, which is to say that users tend to rate books around the same.





6 Hybrid Approach

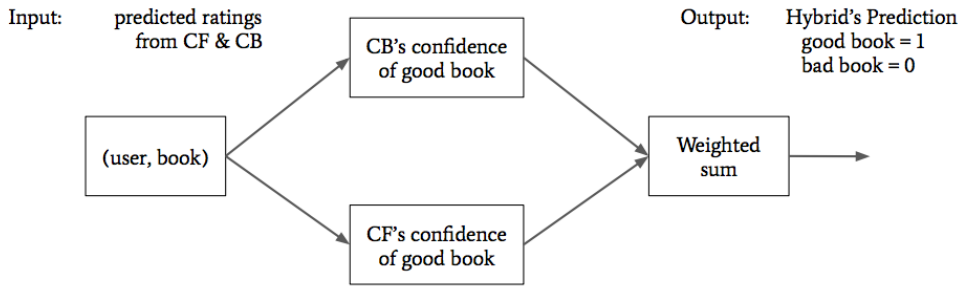
Overview

A combination of CB and CF techniques, referred to as hybrid approach, alleviates some of the problems that each system encounters individually. In fact many successful systems depend on both techniques to make recommendations.

Methods

Our hybrid system works by weighing the likelihood a user will think a book is "good" from CF/CB. It is what is more formally known as a switching hybrid recommendation system.

> "A switching hybrid recommender switches between recommendation techniques using some criteria, such as confidence levels for the recommendation techniques. When the CF system cannot make a recommendation with sufficient confidence, then another recommender system such as a content-based system is attempted." [8]

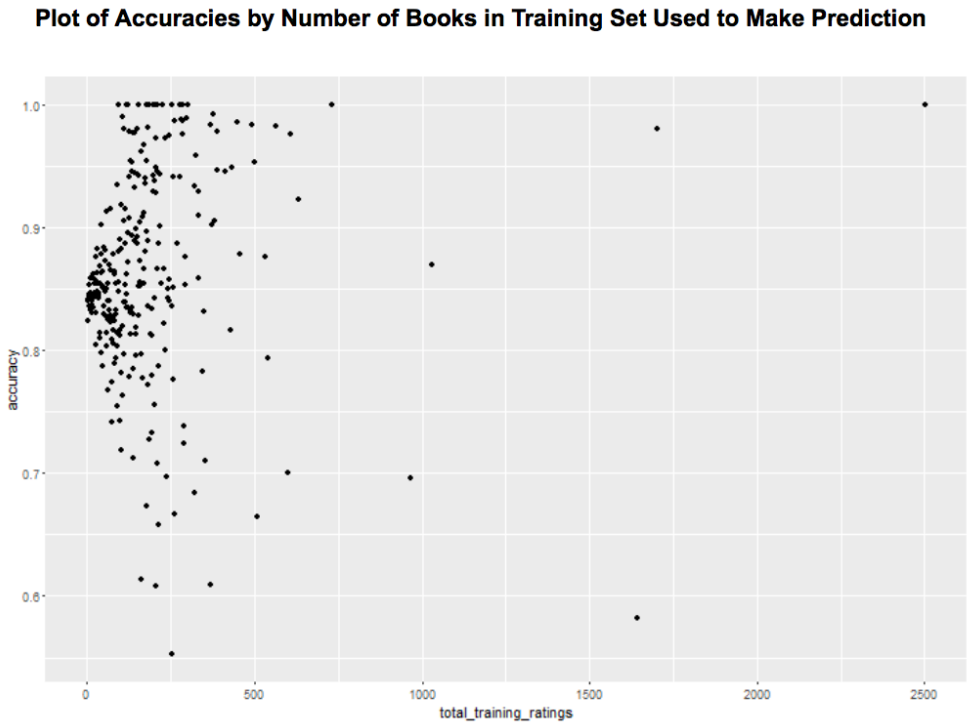


The perceptive reader may have noted that the CB and CF sides output results predicting slightly different things. CB classifies whether or not a user would think a book is good or bad, whereas CF predicts the rating a user would have given a book. Therefore in order to compare the CF to the CB side, the ratings need to be converted to a binary scale (good book, bad book). We did this by setting a rating threshold, to define each category. Recall that our rating scale is 1 to 5. Therefore, it follows that a rating of greater than 2.5 is a good book; any other rating is a bad book. Comparing the performance of each approach on the development data we determine how to weight the predictions. The overall accuracy produced by each approach was comparable, so we've weighed the likelihoods as 50:50.

Results

First, we needed to determine how much to weight each respective component's binary prediction about whether or not the user would like the book. This entailed quantifying how accurate our two systems were as the number of training books used to make those ratings increased.

As we noted earlier, we saw that the collaborative filtering system's recommendations hardly changed as the number of training books increased. Therefore, we chose to use the same weight for all of CF's predictions. Next, we attempted to fit a linear model to the CB side's data to investigate any potential relationships between the number of books in training and the accuracies:



We found that there was not a linear relationship between the accuracies of ratings and the number of books that the classifier used to make its predictions ($r^2 < 0.01$), even after applying log transformations to the data. Therefore, we also chose to give the CB side a constant weight.
Then, we needed to determine how much to weight each system's confidence score. We found that both recommendation systems had accuracies of about 84%. Accordingly, we decided to weight both predictions the same (50/50). We then ran this system on our evaluation data:

Table 2: Hybrid System Evaluation Results

Accuracy	Precision	Recall
84.99%	89.05%	91.68%

Our final prediction system had an accuracy of 84.99%, a precision of 89.05%, and a recall of 91.68%, which outperforms either individual system.

7 Conclusion

All of our systems– purely content-based, purely collaborative-filtering, and hybrid– performed quite well. Looking back on the project, one thing that we might have chosen to do differently in retrospect would have been to spend more time searching for a dataset of ratings with a higher rating variance per user. Had we been able to find such a dataset, our implementations of algorithms would have been tested on data that would have been more representative of what a typical commercial recommendation system could access in creating its predictions. However, given the data that was available to us, as well as the results our various approaches produced, our systems were largely successful, providing insight into how the different systems we regularly use work and the varying algorithms that make that possible.

References

[1] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014.

[2] Bing Liu and Lei Zhang. "A survey of opinion mining and sentiment analysis." In: *Mining text data*. Springer, 2012, pp. 415–463.

[3] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. "Content-based recommender systems: State of the art and trends." In: *Recommender systems handbook*. Springer, 2011, pp. 73–105.

[4] Raymond J Mooney and Loriene Roy. "Content-based book recommending using learning for text categorization." In: *Proceedings of the fifth ACM conference on Digital libraries*. ACM. 2000, pp. 195–204.

[5] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[6] Badrul Sarwar et al. "Item-based collaborative filtering recommendation algorithms." In: *Proceedings of the 10th international conference on World Wide Web*. ACM. 2001, pp. 285–295.

[7] Ana Stanescu, Swapnil Nagar, and Doina Caragea. "A hybrid recommender system: user profiling from keywords and ratings." In: *Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 01*. IEEE Computer Society. 2013, pp. 73–80.

[8] Xiaoyuan Su and Taghi M Khoshgoftaar. "A survey of collaborative filtering techniques." In: *Advances in artificial intelligence* 2009 (2009), p. 4.

[9] Peter D Turney and Patrick Pantel. "From frequency to meaning: Vector space models of semantics." In: *Journal of artificial intelligence research* 37 (2010), pp. 141–188.