Book Recommendations on GoodReads.com

Derrick Isaacson and Abraham Sebastian

Introduction

GoodReads is a social networking website that allows users to share information about books that they are reading, get book recommendations from other users and meet new people online.

A GoodReads user maintains bookshelves of books that he has read or is currently reading. He can rate books that he has already read and can also write reviews for books. Users can find friends within GoodReads and compare books with them. Users can also recommend books that they have liked to their friends.

We propose to create an automatic recommendation system that uses prior knowledge of a user's rating for some books to suggest new titles to add to his collection. We would like to make recommendations with high precision, i.e. there must be a high probability that the user would like the suggested books. We use user-based approaches where we use similarity measures among groups of users to make recommendations. For example, we use K nearest neighbors clustering to group users together based on the books on their bookshelves and their personal ratings for those books. We predict ratings for a book that a user has not read from the rating given to that book by other users in his cluster. We also use Item-based methods where we create networks of related items to make recommendations. So in this case a recommendation model is not tailor-made for a particular user but is instead based on the particular items he selects.

Data Acquisition

To recommend books we needed data on users and the ratings they gave to books. We acquired the data by querying GoodReads' public web api.

GoodReads organizes their data by users and "bookshelves". A user has multiple bookshelves that each contain multiple books. There is a standard set of bookshelves which every user has: "read", "currently-reading", and "to-read". Most users stick to the standard set, but GoodReads provides the ability to create custom bookshelves. This ends up acting like a form of tagging. For example, the play Macbeth by William Shakespeare has been put on a "plays", "Shakespeare", or "classics" bookshelf by multiple users.

We acquired our data mainly through the "get the books on a shelf" method. Given a user and bookshelf name we queried for all of the books. This provided several fields of data for each book on the shelf such as user rating, average rating, and catalog data such as title and ISBN.

GoodReads does not expose a list of user IDs so coming up with a list of users to get data for was a challenge. We found that if we took the user IDs for our own profiles and queried for numbers nearby, that we were able to find a sufficient number of active users. Because we were interested in books that users have already completed and given a rating too we used the standard "read" bookshelf for each user. This was still challenging because many

people list their bookshelves as private, and many people do not have any books on their "read" shelf. We ended up having to query for about 10 times the number of user IDs as we actually needed data for in order to find enough public bookshelves that have books on them.

Model Evaluation

Similar to Facebook Friend Suggestion [3], we would like to predict a rating that a user would give each book if they were to read it. GoodReads allows you to rate books on a scale of 1-5 stars with 5 being the best. However, we do not simply want to calculate how many stars a user would give a book, but find the top X number of books that they would like. So we would like to calculate these numbers on a continuous scale and pick the books with the largest X values, rather than simply classify the books for the user.

There is also a challenge in testing our hypothesis once we calculate it. Because it is not possible to make the recommendations and then have the users read the books and get back to us in a timely manner, we decided to use some form of hold out cross-validation. As we prototype our system and iterate we plan to experiment with different forms of cross-validation to see what our needs are. Although we cannot recommend books and then test it out on the users, we *can* obtain data for a fairly large number of users so a simpler form of hold out cross-validation will likely be sufficient.

Item-based methods

Similarity Computation

Item-based collaborative filtering finds relationships between pairs of items while ignoring similarity between users for simplicity. It performs regression that is simpler than linear regression. Rather than computing f(x) = ax + b, it calculates f(x) = x + b for each pair of items.

The input we use for computing item similarity is a matrix of users and their ratings for specific items. We compute a similarity measure between pairs of items as follows. We derive vectors i and j for the two items using the ratings of all users that have rated both items. For example:

	i	. j
u1	4	2
u2	5	5
u3		3
u4	2	

In the above table user u3 has not rated item i and user u4 has not rated item j. So we have $i = [4\ 5]$ and $j = [2\ 5]$. We then compute the average vector from i to j as ((2-4) + (5-5))/2. We use this to measure the similarity between two items.

For a given item j that a user has not rated and some existing ratings for a user we can

predict how a user will rate j. The prediction is made by adding the average vector from i to j to the user's rating for i, then taking the average over all i.

We also followed one common improvement to this algorithm by weighting the different i items used to predict the new j item. We give a higher weight to i when more users have rated both i and j.

Learning performance

Slope one is used because

- 1. It is simple to implement and does not require much tuning.
- 2. It avoids overfitting the training data because of its simplicity.
- 3. It is relatively fast to compute.

We implemented Slope One in C# and ran it on a machine with a 2.08 GHz AMD Athlon processor, 1 GB of RAM, running Windows XP and Visual Studio 2008. The one major performance bottleck that we ran into was the n^2 amount of memory that the similarity calculations require from calculating all pairs of items. Because not every pair of items have been rated by some user we represent the matrix as a dictionary of dictionaries where the keys are ISBNs for the books. Even then we could only compute the matrix for 4000 users before running out of memory on our machine.

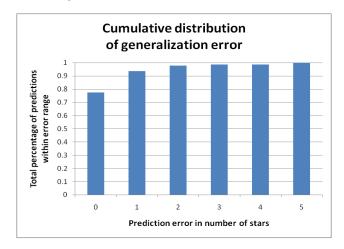
Future work could improve this a few ways. One would be to not store all of the data in memory at a time. A second would be to cut the size of the matrix in half by only storing vectors in one direction between a pair of products. The vector from item j to item i is simply the negative of the vector from item i to j we could avoid the storage overhead and do the reverse lookup in our table when necessary. However, we did not see much improvement when increasing the number of users from 2000 to 4000 so we expect the results to not change much.

Recommendation performance

We tested our algorithm through cross validation. First we held out of our training set 30% of our users. After training the algorithm we made predictions using hold-out one cross validation. We did this by repeatedly holding out a single book the user had rated and making a prediction on that book looking only at the other ratings the user had given. We tried a few methods of generalization error to find the best indication of how this would perform as a recommendation system. First we calculated the average difference between our prediction and the actual rating. This gave us an error of about 0.7 out of 5 stars.

As mentioned above, a recommendation system is only interested in books for which it predicts maximum ratings. We improved our error measurement by only considering 5 star predictions. After tuning some parameters of how the algorithm rounds its predictions we were able to predict exactly the user's rating about 78% of the time and within 1 star 92% of the time. As a comparison, only 35% of the ratings are 5 stars so our predictions are actually meaningful.

Below is a graph of the generalization error, where the error is reported as percent of predictions within an error range from 0 to 5 stars.



As referenced above, there was one rounding parameter we could tune in the system. We discovered that adjusting it gave different results and after looking into it we discovered we were trading off precision for recall of the 5 star ratings. Below is a graph of the tradoff as a function of the rounding parameter. We tuned the system for precision over recall. The algorithm returns hundreds or more of 5 star predictions so we did not need high recall to recommend a few good books to the user.

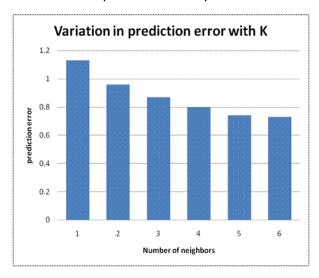


User-based methods

User-based collaborative filtering works by looking for users who share the same rating patterns as the active user (the user whom the prediction is for). These methods work by first clustering together users who are similar and then deriving the rating for a new item from the ratings given by other users in the cluster. We used K nearest neighbors to cluster users together. For a dataset with N users and M book titles, we implicitly represent each user as a point in an M dimensional plane, where each coordinate is the rating for the book corresponding to that coordinate axis. We then compute the similarity measure between a

pair of users as the average of the squares of the difference between ratings for books that they have both rated.

We predict the rating for a book that a user has not yet rated by computing the weighted average of ratings given to the book by other users in the same cluster, where the weights are the similarity measures between pairs of users. To determine the optimal value of K, we ran our algorithm on the dataset using different values of K and tested our generalization error with hold out cross validation. The graph below shows the average difference between our predictions and the actual ratings given by users for progressively higher values of K. We see that our predictions do not improve much beyond K=5.



References

- [1] Daniel Lemire and Anna Maclachlan. Slope One Predictors for Online Rating-Based Collaborative Filtering, In SIAM Data Mining (SDM'05), Newport Beach, California, April 21-23, 2005.
- [2] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. *Item-based collaborative filtering recommendation algorithms*. In WWW10, 2001.
- [3] Eytan Daniyalzade and Tim Lipus. *Facebook Friend Suggestion*. In CS 229 Machine Learning Final Projects, http://www.stanford.edu/class/cs229/projects2007.html, Autumn 2007