Book Recommendation

(with collaborative filtering, clustering, and natural language processing)

Introduction

A personal challenge I've encountered is finding the next book I would like to read, and I've often wished for a recommendation engine that would suggest books that are similar to books that I like. This could come in the form of a collaborative filtering system, where recommendations are based on ratings from other users, or similarities in book metadata that would not depend on other users. This project is my attempt to build that engine. Beyond my personal use, this recommender could be useful to an online book retailer to help drive sales or to a social network for readers which could help categorize books without the need for expert labeling to sort genres.

To accomplish this, I'm using data from bookcrossing (http://www2.informatik.uni-freiburg.de/~cziegler/BX/), which features ratings for 183824 books (listed by ISBN) from 77553 users (given by anonymized userIDs). The dataset includes a separate file with book information such as the title, author, and publisher of the book.

Data Wrangling

I started my data processing by dropping ratings of zero from the ratings dataframe. Those ratings were meant to be implicit from the text review given by the user, but I didn't have access to those reviews, so those ratings would not be useful for what I'm trying to accomplish.

The ISBN column in the ratings dataset needed a lot of work. That information was input by users, and many of the entries were not uniform like we would prefer for an algorithm. Many users left off leading zeros, trailing characters, or didn't know where to find the ISBN on the book they read so they input all zeros or another number from the back of the book. To fix this, I cross referenced the ISBN in the ratings dataframe with the ISBNs in the books dataframe using pandas.str.contains() to see if I could get a partial match. For example, ISBNs are standardized to have either 10 or 13 digits, so if I found a 9 digit ISBN in my ratings data, I tried to match it to 9 of the digits in a 10 digit ISBN in the books data, and if that was successful, I updated the ratings data with the correct ISBN.

	User-ID	ISBN	Book-Rating
55	276762	N3453124715	4
247	276856	20103389	0
336	276875	273755	7
337	276875	14366020444	8
368	276875	88741800047	8

Table 1: None of these are valid ISBNs. There's thousands of these.

During the above process, I discovered that many of the books in the ratings data had valid

ISBNs, but were not listed in the books data. I needed to fill in those gaps, so I used goodreads' API to get book details by ISBN, then made new entries in the books dataframe with that data. The goodreads API also returned the book's description as it would appear on the back of the book or on a webpage for the book, so after making sure as many books from my ratings data had a corresponding entry in the books data, I used the ISBNs in the books data to get a description for any books that had one available from the goodreads API. I used the beautifulsoup library to strip any HTML from those descriptions and stored that string on the books dataframe.

	Author	Title	ISBN	Publisher	Year	Description
0	Mark P. O. Morford	Classical Mythology	195153448	Oxford University Press	2002	Featuring the authors' extensive, clear, and f...
1	Richard Bruce Wright	Clara Callan	2005018	HarperFlamingo Canada	2001	NaN
2	Carlo D'Este	Decision in Normandy	60973129	HarperPerennial	1991	Here, for the first time in paperback, is an o...
3	Gina Bari Kolata	Flu: The Story Of The Great Influenza Pandemic...	374157065	Farrar Straus Giroux	1999	The fascinating, true story of the world's dea...
4	E. J. W. Barber	The Mummies of Ürümchi	393045218	W. W. Norton & Company	1999	NaN

Not all books had desctiptions available

Collaborative Filtering

I started with the standard question for a recommendation system: 'Users who liked this book also liked what other books'. I did this by:

1. Filtering the ratings data by that book's ISBN to get a list of userIDs for users that had rated the book.
2. Filtering the ratings data by that list of userIDs to get all the books those users had read
3. Creating a pivot table with userIDs as rows and ISBNs as columns.

The resulting table ends up being incredibly sparse, since most users have not read most books. I was concerned that, if a book only had one rating of 10, it would appear at the top of a list of recommendations above a book that had four 10s and a 9, which didn't seem appropriate. To combat this, I used the following formula

$$\frac{\sum_{i=0}^{n}(rating_i * originalbook_i) + 5.5\text{k}}{\sum_{i=0}^{n} originalbook_i + k}$$

which had the effect of initializing each rating with k 'dummy' ratings of 5.5. This prioritized books

that had many high ratings over an entry with on rating of 10. I multiplied the rating by the rating each user gave the original book, since I wanted ratings given by users who liked the book we're comparing against to count more than users who disliked it. If I'm searching for recommendations because I liked a book, I care more about the opinion of people who rated that book a 10 than I do about the opinion of people who rated it a 1.

The natural extension of this method is to apply it to all books a user has rated instead of just one book. The method was similar:

1. Get a list of all books rated by a given userID
2. Filter by that list of books to get a list of all users that have rated those books
3. Create a pivot table

To weight the scores of other users based on their similarity to the user in question, I gave each user a similarity score based on the RBF

$$Similarity = \sum_{i=0}^{k} .95^{(x_i - y_i)^2}$$

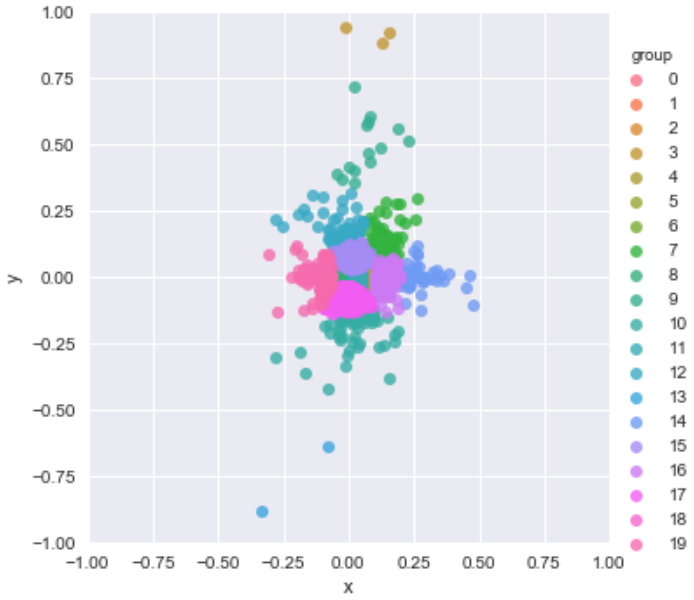
Then each book had a score calculated by using similarity in the weighted average as before:

$$\frac{\sum_{i=0}^{n} (rating_i * similarity_i) + 5.5\text{k}}{\sum_{i=0}^{n} similarity_i + k}$$

Clustering

One of the shortcomings of the above collaborative filtering methods is that in restricts what books can be suggested to books that have been read by people who have read the same books as the user in the query. I tried to address this by using K-means clustering to create larger neighborhoods, where readers who like a specific genre or style of book would be grouped together even if they hadn't read the same exact books.

Before trying to cluster, I used sklearn.decomposition.TruncatedSVD to reduce my vector space down from 180000+ down to something more manageable. I found that I had the best results with only 2-5 vectors after reduction, since with any more than that the neighborhoods got to spaced out to form meaningful groups, and I would end up with all but one cluster with only one member, with the remaining 70000 users in the same cluster. As mentioned before, most of the users in the dataset have rated very few books, so with SVD, most of them ended up near the origin, with the more prolific readers being so spaced out they get their own cluster. Having each user rate more books improves the results. As it is now, members of this largest cluster tend to get recommendations for all around popular books like Fahrenheit 451, Lord of the Rings, and To Kill a Mockingbird, which isn't a bad set of books to start suggesting.



I like to call group 11 "Harry Potterville"

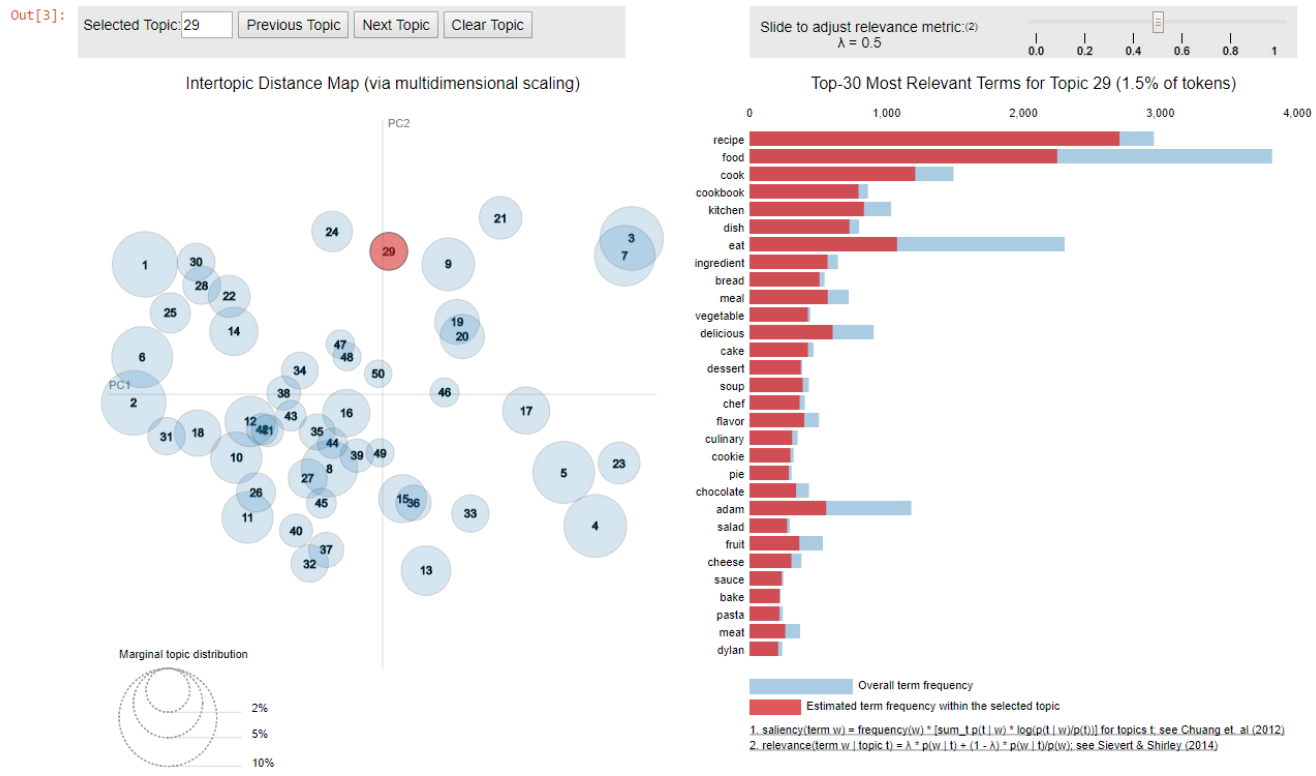
NLP on Descriptions

The methods above do a respectable job of generating relevant suggestions, but they fall victim to the cold-start problem in recommendation engines. Many of the books in the dataset have no ratings, and would not show up in any results. Those books would also not be able to be used as the seed for generating suggestions. To help populate the results with books that have no ratings, I decided to take the descriptions I gathered and perform some language processing to find books about similar topics or genres.

I started by filtering out any books where the description wasn't in English using the langdetect library. In previous methods, this hadn't been a problem (users who read German language books were clustered with other German readers automatically) but for language processing, having descriptions in dozens of languages was an issue.

Using the spacy library, I lemmatized the descriptions to remove any grammar and punnctuation. This it so words like “walked”, “walking”, and “walks” all are transformed to “walk”, helping to standardize the vocabulary of our text. I then used gensim.models.Phrases to join words into bigrams when they appear together frequently, turning 'award winning' into 'award_winning' and allowing it to have its own representation in the bag of words model independent of the other two words. I ran this process twice, generating three word and four word phrases. These lemmatized words and phrases built the vocabulary for the bag of words model. After removing stopwords using spacy.en.STOP_WORDS, I had a list of keywords for each description that I could analyze.

Those keywords opened up two pathways to pursue. The first was latent dirichlet allocation, where I used gensim.models.LDAmulticore to generate 50 topics. Examining these topics shows some to be about types of books (such as textbooks), genres (such as murder mystery or fantasy), or style (focused heavily on character names and actions vs being about the author's tone and themes).



Topic 29: Cooking

I used the topic model to make a 50 element vector for each book, and then used cosine similarity to find other books on similar topics. Those results were sorted on the strength of the relationship.

The second pathway I pursued was using word vectors. Using gensim.models.Word2Vec, I generated a 100 element vector for each word in the document's dictionary. I then summed the vectors for the words in each book's description to make a vector for the document as a whole, then used cosine similarity to compare books. One interesting advantage of this method is the description does not have to be from a book that's actually in the dataset. Feeding in the text “romantic pirate adventure” or “african poetry” will yield relevant results.

Recommendation for use and future study

The collaborative filtering methods used here had a heavy bias towards frequently rated books, leading to many users having a Harry Potter book in their top 10. Suggesting books that have wide, cross-genre appeal is not necessarily a bad thing, but it doesn't do as much for people who are trying to 'discover' lesser known books they might enjoy. In contrast, the language processing methods place no emphasis on how popular a book is, since part of the motivation for using that method was a lack of ratings for some books. The best implementation of a recommender is probably some hybrid of those two, giving users a variety of options.

The language processing methods only currently work in English. Spacy has support for French, German, and Spanish, potentially allowing the model to be expanded to include more of the users in the dataset, but since I don't understand those languages, I wouldn't be able to interpret or judge the results.

Finally, the clustering method would likely perform better with manifold learning instead of SVD so it could form nonlinear groups, but I ran into memory problems due to the size of the dataset. There are libraries available to handle those problems with sparse input, but I struggled and ultimately failed to get them working on my Windows machine, since the developers seem to anticipate people working on these problems in Linux or Osx.