DATA DRIVEN BUSINESS

DDB assignment
Recommender systems

Recommender systems retrieve and rank information from content-serving websites like Netflix, Amazon, Spotify, etc. and their interaction with its users. The objective is to develop a book recommender system using collaborative filtering techniques. This assignment is divided into two parts: the basic tasks, which account for 65% of the grade, and the advanced tasks, which make up the remaining 35%.

Data

This dataset contains six million ratings for ten thousand most popular (with most ratings) books from the Goodreads datasite. There are also:

- books marked to read by the users
- book metadata (author, year, etc.)
- tags/shelves/genres

Files

- ratings.csv contains time-sorted ratings:

  user_id,book_id,rating

  1,258,5

  2,4081,4

  2,260,5

  2,9296,5

  2,2318,3

  Ratings range from one to five. Both book and user IDs are sequential, with books ranging from 1 to 10000 and users from 1 to 53424.
- to_read.csv lists books marked as "to read" by users, in user_id,book_id pairs, sorted by time, with nearly a million entries.

- books.csv includes metadata for each book (Goodreads IDs, authors, titles, average ratings, etc.), extracted from Goodreads XML files available in books_xml.
- book_tags.csv contains user-assigned tags/shelves/genres for books, represented by their IDs and sorted by goodreads_book_id in ascending order and by count in descending order. Each tag/shelf is assigned an ID.
- tags.csv translates these IDs to names. There are many tags as these are user inputted. For example: Read, Currently Reading, Want to read, DNF

Each book may have multiple editions. goodreads_book_id and best_book_id typically refer to the most popular edition, while goodreads work_id refers to the book in an abstract sense.Goodreads book and work IDs can be used to create URLs as follows:

- https://www.goodreads.com/book/show/2767052
- https://www.goodreads.com/work/editions/2792775

Note that book_id in ratings.csv and to_read.csv maps to work_id, not to goodreads_book_id, meaning ratings for different editions are aggregated.

Basic tasks

Pre-processing of the user-book information

Get a feel of the data. Describe characteristics of user rating data (e.g. verify the typical long-tail distribution of the ratings). Report your favorite stats that can help you guide the design of recommendation systems. Organize and clean books with all their metadata and user-item ratings.

Implement a item-based and user-based collaborative filtering approach

1. Obtain the user-book matrices, compress the size of the matrix (# users, # books) using the previously derived statistics (e.g.) and use a K-Nearest Neighbor CF approach to recommend books to particular users (user-based) and find similarly rated books (item-based). Fit the model and make recommendations for these scenarios, validate these predictions qualitatively. S*klearn* implements the aforementioned methods.
2. How do you deal with missing values in the user-book matrix? Describe under what situations this approach is likely to fail. What happens with lesser known books? Does this method scale well?

Implement a matrix factorization collaborative filtering approach

1. Obtain the user-book matrix and perform SVD based matrix factorization on it (*hint*: certain scipy linear algebra package can be handy). Matrix factorization is the process of breaking down one matrix into a product of multiple matrices. It's extremely well studied in mathematics, and it's highly useful. The decomposition is written as $R=U\Sigma V^T$ where R is the sparse user-rating matrix, U is the user features matrix , V^T is the book features matrix and Σ is the diagonal matrix of "weights". To get an approximation of R, we take these matrices and keep only the top k features. Be careful, this step can be computationally demanding for large matrices! What do you observe?
2. With the predictions matrix for every user, we can build a function to recommend books for any user with only matrix multiplications! Figure this out. All I need to do is return the books with the highest predicted rating that the testing or query user hasn't already rated.
3. Compare this approach to the item based one and describe in what situations this approach is likely to fail. For example, (A) How to deal with new users and items, and (B), how to deal with new interactions (e.g. ratings, clicks, etc.) and other scenarios that may arise in real-time production systems. Do these algorithms offer an online version?

Advanced tasks

`Pick at least one! Although you can combine. Other ideas are also admitted upon request.`

- Create and train/test split on the user rating data to evaluate the predictions made by the built recommender system. Use metrics such Mean Absolute and Square Error (MAE/MSE) to assess the quality of the predicted ratings. Select user subset that have at least a certain number of ratings across books. Additionally, optimize the number of "features" on a held out validation test according to the quality of the approximation.
- Scrape some additional user or book data from the web. As a clarifying example, grab Amazon reviews for this subset of Goodreads books. Go obtain book reviews from other sources. Combine the ratings/reviews and analyze their distributions. Alternatively, apply a simple EM pipeline to automate the addition of new books to the original Goodreads database.
- Visualize book factor data (from the matrix factorization block) and match it accordingly with some well known features such as genre, theme, popular tag…. You may cluster or reduce the dimensionality of similar book tag data into few categories or buckets (or just simple genre classification). Finally, you can try to see if the SVD latent item features correspond to these categories by using visualization methods.
- Implement a content based filtering approach using NLP. You may use the shelves/tag or the data to compare books or find alternative descriptions of the book database. Recall that you can

generate book features by constructing tf-idf vectors of the book descriptions, tags, and shelves. Come up with a hybrid recommendation engine to palliate some of the problems of collaborative filtering.

- Use the *to_read.csv* bookmark data as implicit rating data to enhance or complement the collaborative filtering method.
- Use an alternative recommender system library (e.g. Surprise) or algorithm (e.g. Alternating Least Squares) on the basic tasks.
- Develop a local web-app that deploys the implemented recommender system. Implement a fuzzy matching query system to search books/authors in your engine (i.e. *Scot fitzgerad* for *F.Scott Fitzgerald*)

Delivery format

Upload a python notebook or script with the required tasks on Canvas. Additionally, you can also upload (PDF, word,etc) explaining some part of the approach or comments and observations of different experiments.