Recommender System Report

Problem Context and Identification

Context

Recommender systems are widespread in all aspects of user-facing industries, from Netflix, to Amazon, to LinkedIn or to YouTube. Due to the large volumes of content, products or posts being generated daily, users would be overwhelmed with information and simple searches may not generate adequate results that may be relevant to the user.

As a result, recommender systems came about as a subclass of information filtering systems that provide suggestions for items that are most pertinent to a particular user. For example, over 70% of watch time on YouTube is spent watching videos the underlying recommender algorithm suggests.

The goal of this project is to develop a recommender system designed for Amazon shoppers that would recommend books they would most likely enjoy based on the reviews of others.

Problem Statement

How can we accurately recommend 5 new books to a user using an Item-item approach to encourage expanding user preferences to books they may not have otherwise found on their own?

The Data

The data for this project comes from here and provides two datasets to work with. The first dataset has rows of information in the form of `bookID, userID, rating, timestamp`, representing a given user's rating for a given book. This is generally the expected format for collaborative filtering algorithms. This dataset contained about ~51M rows of data.

The second dataset contains metadata for all books in the first dataset and has columns like: title, description, price, category, etc. for each book.

Exploratory Data Analysis (EDA)

Exploratory Analysis

Initial exploration of the dataset showed that there were >51M rows of unique reviews in the dataset with 4 features representing user ID, book ID, rating and timestamp.

How many unique users and books were in the dataset?
It was determined that there were 2,930,451 unique book IDs and about 15,362,619 unique users.

What is the average number of books a user rated?
On average, it was noted that each user rated about 3.34 books.

What is the average number of ratings a book received?
On the other hand, it was noted that each book received about 17.51 unique ratings on average.

Data Wrangling

For the most part, the dataset was in the appropriate format required for collaborative filtering algorithms. However, substantial wrangling was required to both downsample the data to a size that wasn't computationally expensive, and drop null values from the dataset.

The initial dataset prior to downsampling contained about 6.7M rows with missing data. Since this represented only about 13% of the dataset, dropping the missing rows still yielded a large dataset.

Due to the size of the dataset, the PySpark library was used to take advantage of its lazy evaluation and parallel processing to downsample the dataset to about ~32K rows -- this size limit was determined iteratively as the max that the `Surprise` library could handle for later steps.

The method used to downsample the dataset to a representative size was to go based off of the number of unique book IDs, and not unique user IDs. Since each book received about 17.51 reviews on average, compared to each user only giving about 3.34 reviews on average, downsampling the unique book IDs from ~2.9M to about 1,860 yielded a dataset of ~32K rows of reviews. This worked well for a representative sample for our model.

Finally, the spark DataFrame was converted to an intermediate pandas DataFrame from which the Surprise library had a function to convert to a Surprise dataframe. The rating scale did need to be specified, which was on a scale from 1.0 to 5.0 stars.

Modeling and Evaluation

The Surprise library provided several flavors of collaborative filtering algorithms to work with falling in distinct categories such as matrix factorization, k-NN inspired, Slope One and Co-clustering.

Baseline Models

Several baseline models were built to determine the best performing baseline algorithm. Of the options, the SVD, NMF, KNNBaseline, KNNWithMeans, KNNBasic and SlopeOne algorithms were tested.

Using the MAE and RMSE as metrics, the top two performing baseline models were the SVD and KNNBaseline models with an MAE score of 0.712 and 0.713 and RMSE score of 0.967 and 0.966, respectively. This meant that on average, the top model's predicted rating for a given user on a given book was off by about 0.71 stars, which wasn't too bad.

Model Optimization using GridSearchCV

Since the size of the dataset was now not really large, Grid Search was chosen as an exhaustive approach to determine the best hyperparameters for the top two models.

For the SVD model, hyperparameters such as number of epochs (the number of iterations of the underlying SGD procedure), lr_all (learning rate for all params) and, reg_all (regularization term for all parameters) were chosen. The combination of parameters that gave the best MAE score was 200, 0.002 and 0.1, respectively. This yielded a model with an MAE score of 0.710 and RMSE of 0.965.

For the KNNBaseline model, since this is a k-NN inspired algorithm a dictionary of options for similarity measure was provided for tuning which contained parameters for similarity metrics like msd and cosine as well as min_support (the minimum number of common items for the similarity not to be zero). Additionally, a dictionary (bsl_options) provided options for configuration which included whether the underlying algorithm was ALS or SGD, and whether the regularization parameter of the cost function was 1 or 2. The combination of parameters that gave the best MAE score was the ALS algorithm using the msd similarity metric with a min_support of 5 and regularization parameter 1. This yielded a model with an MAE score of 0.713 and RMSE of 0.966.

Finally, based on the results of the tuned models, the tuned SVD model was chosen as a basis for the recommendation system since it slightly outperformed the KNNBaseline model, and k-NN models tend to become computationally expensive as the size of the dataset grows.

Determine The Top-N Recommendations for a Random User

After selecting the best performing model, the next step was to use the model to predict the top 5 book recommendations for a user based on other books they liked.

The first step in doing so was to predict ratings for all pairs (user, item) that were not in the training step. This step was very computationally expensive since the relationship between the predicted feature space and the size of the user-item pairs was quadratic. Therefore, a large dataset would require a very expensive computation to yield the feature space of predicted ratings for all pairs.

This first step yielded a dictionary where keys are user (raw) ids and values are lists of tuples: [(raw item id, rating estimation), ...] of size n. In essence, it returned the book IDs of the top-n recommended books for all users. This allowed for easy lookup by user ID to get a user's top-n recommendations.

Since the result of the previous step just yielded a user ID and N-recommended book IDs, the results weren't meaningful. So the last step was to use the returned recommended book IDs to get metadata about each book from the metadata dataset like title, description, price, etc. for more meaningful results.

The final product was a pandas DataFrame showing the top 5 recommendations for a user, along with the recommender system's predicted user rating, and metadata for each book (see Figure below).

	title	price	description	books	predicted_ratings
0	Army Field Manual FM 22-100 (The U.S. Army Lea...	Price Not Available.	No Description Available.	1420928244	5.0
1	The Essences	$23.99	Writing this book was a very fun process for m...	1514491877	5.0
2	Winning the Staffing Sales Game: The Definitiv...	$19.99	No Description Available.	1543461468	5.0
3	Wolfbaene	$7.81	My name is Michelle Dennis. I live in Ellenbro...	1849633436	5.0

Figure 1: pandas DataFrame showing the top 4 recommended books for a random user, along with the recommender system's predicted ratings and useful metadata for the books.

Conclusion

In the end, the goal of the project was successfully achieved and we were able to predict the top 5 recommended books for a given user to expand their preferences. The underlying algorithm for the recommender system was a matrix factorization-based SVD algorithm that had quite decent performance with only a MAE error of 0.71 stars on average.

Due to the very computationally expensive step that required predictions for all user-item pairs in the dataset, the recommender system would not scale well as the size of the dataset increased. Other efforts would need to be taken to rebuild the recommender system more robustly using libraries that offer more options for determining a user's top-n recommendations, perhaps like Tensorflow's recommenders library.

Lastly, the final output of the system was a simple pandas library containing the top recommended books for a user, along with the system's predicted ratings and some useful metadata. Some interesting next steps for the project may be to create a simple API endpoint for the recommended books along with the metadata as a means to present that information in a more useful way in a Flask app for user interaction.