Books Recommendation System using Collaborative Filtering

Abhinav Chand, Quynh Do and Bijan GURUNG
STAT 766 (Fall 2022)

1. Background

Recommendation system started to gain traction after online shopping or e-commerce took the center stage in marketing and a huge dataset started to become available in this field. Recommendation system relies on content-based filtering, collaborative filtering or the hybrid model. As the dataset became humongous, the method employed neural networks in collaborative filtering. The system makes use of the latent features (implicit or explicit) in matrix decomposition or factorization and in turn, predicts the item rating and item preference by a user. Recommendation lists are presented after analyzing user preferences, items and their features, user-item interaction or preference data, and other associated information (timestamp, spatial data). We took the task of analyzing books rated by users (from Amazon) for our recommendation system. There are three different types of datasets, namely users data, books data, and ratings data (source: https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset).

2. Motivation and Goal

Thousands, if not millions of books are picked or purchased by clients or users everyday in the online shopping sites (e.g. Amazon). Users are requested to voluntarily rate the books and are also requested to provide some of their basic information such as preferences, age, etc. Rating data and the users' information (also books' features) constitute the key part of content-based filtering as well as collaborative filtering. Our goal is to employ collaborative filtering in defining the latent features which will then be used to predict the missing rating of existing or new users. Parameters' tuning will be carried out for the appropriate model for features matrices. Once the ratings are predicted, the related books are listed for recommendation for the particular user.

3. Dataset (EDA)

The `'~/Ratings.csv'` is a file of shape `(1149780, 3)`. The columns are `'User-ID'`, `'ISBN'`, and `'Book-Rating'`, where `'ISBN'` contains alphanumeric characters representing each book. Books are rated from `0` to `10` where `0` represents no rating. A large part of the data, i.e. `716109` rows contain `0` rating. There are `105283` unique users and `340556` unique books. We dropped `0's` from the data and visualized the books that received a high number of ratings. The corresponding `'ISBN'` numbers showed the information about these books from the `'~/Books.csv'` file.

Pre-processing was carried out prior to the matrix factorization algorithm. We removed `0` ratings from the `'~/Ratings.csv'` and split it into a train and a test set. The details of this procedure can be found in `train_test_split.ipynb`.

4. Methodology

I. User Based Collaborative Filtering:

In order to predict the missing ratings for the users in the test dataset, we wrote a function called `rating(user_id, isbn, k)` where the `user_id` is the id of the user whose rating we require, `isbn` is the id of the book whose rating we require, and `k` is the the number of neighbors for KNN.

The aforementioned function finds the k nearest neighbors of the given `user_id` based on cosine similarity of the user vector, then it takes the weighted average of the ratings of the neighbors based on the similarity values. This is the predicted value for the missing rating.

Note: The details of this function and examples related to it can be found in the `Cosine Similarity.ipynb`.

II. Item Based Collaborative Filtering:

This is one simplest method for finding out item-item similarity coefficient. The `DataFrame` containing the Ratings (`'~/Ratings.csv)` was converted into a `DataFrame` representing a matrix of `'User-ID'` and `'ISBN'` by using the `.pivot()` function. The columns are named with `'ISBN'` and the indices with `'User-ID'`. The matrix shown in the `DataFrame` was transposed and stored in a 2-D array. `TruncatedSVD()` was applied to reduce the dimension to 12 (features) and fitted into the transformed matrix. The correlation coefficient was calculated amongst the `'ISBN'` with `np.corrcoef()`.

The indices of books (`'ISBN')` were stored in a list. Correlation of a book of interest to the rest of the books was carried out and the top most highly correlated books were presented with correlation coefficient `>0.9`. Information about these books were displayed from a `DataFrame` containing books (`'~/Books.csv'`). This method is mostly applicable in a 'cold start' instance when the user rated just one or two books with non-zero rating. There was a memory issue when the correlation was conducted in the whole dataset and the result was stored in a 2-D array format. So, we sampled the rows from the `DataFrame` containing Ratings (`'~/Ratings.csv)` and carried out the analysis for highly correlated books to a book of interest. As a caveat, if the book does not receive a single rating, this function does not work. So, the book should receive at least one non-zero rating in order for this function to perform. The details of this process is found in `truncSVD_books.ipynb`.

III. Matrix Factorization:

The function that calculates the matrix factorization of a given matrix is called `MF(M,k,max_it,lambda,mu)`. This can be found in both the `MF evaluation.ipynb` and `MF validation.ipynb`. This takes in the required matrix M, the number of latent features `k`, maximum number of iterations `max_it`,

regularization parameter for U called `lambda`, and the regularization parameter for V called `mu`. Our algorithm uses `PyTorch` and optimizes the gradient descent using `ADAM`. We experimented with the various schedulers in `Pytorch`, but it reduced convergence time. At the end, we chose not to use any scheduler.

For hyperparameter tuning, we took the ratings of the 500 users that rated the most number of books and sampled 20% of that data for validation. Then we changed the ratings from the validation to 0 and combined everything to create a matrix. Then we tested our matrix factorization prediction on the validation for the different values of `k: [5,10,20,100,200], lambda:[0.01, 0.1,0.5], mu:[0.01,0.1,0.5]` . We concluded that `k=10, mu=0.1, lambda=0.1` were the best parameters. Details of this procedure can be found in the `MF validation.ipynb`.

For inference, we realized that predicting all the test values at once was not feasible. We observed that there were about 20000 unique users in the test dataset. We decided to predict the ratings in batches. We divided the test data into 20 batches so that each batch would consist of around 1000 unique users. Then we combined each batch with the ratings data of the 1000 users that have rated the most number of movies. We then performed matrix factorization on this combined matrix and calculated the `MSE` for the required entries. The details of this procedure can be found in the `MF evaluation.ipynb`.

5. Results and Discussion

The `RMSE` from the matrix factorization model was 2.7. We realized that our matrix factorization algorithm takes a very large number of iterations to converge. If we had enough time and computing power, we could run longer iterations and improve the result. We also believe that we can tweak our algorithm to converge faster. We did experiment with `ADAM` and `SGD` with momentum, but we could try other optimization techniques, and play with the learning rate and other parameters for gradient descent algorithms. We did also experiment with different schedulers in `PyTorch` but that did not improve the rate of convergence.

For future work, we can split the original dataset into a train and test set such that we can apply the user based collaborative filtering and compare it with the matrix factorization model. We could also make comparisons with `truncatedSVD` with the Pearson correlation model, especially for users with few ratings.

Due to time constraints, we could not apply neural collaborative filtering to our dataset. We hope to create these models in the future and compare them with our matrix factorization model.