DSGA-1004 Final Paper: Recommendation System

Outline

Basic Recommender System

Data Splitting and Subsampling

In the data splitting process and subsampling, we followed the directions on project description and added our own implementation based on the understanding of the dataset as well as concerns about dumbo capacity. Our detailed approach is as follow:

1. Subsampling dataset into 2%, 10%, 25% of total dataset for baseline fitting and tuning hyperparameters
2. Do train/test split following
    - training data = 60% of users with all interactions + half of the interactions for the 20% users in validation set and 20% of users in test set
    - Validation data = 20% of users with 50% interaction
    - Test data = 20% of users with 50% interaction
3. During the preprocessing period, we also filtered out users with less than 10 interactions

Model fitting and prediction

Baseline model

We first fit our baseline model with 2 % of data and default value for hyperparameters. The detailed values for hyperparameters are: rank=20, maxIter=10, regParam=0.1, alpha=1

The baseline performance with evaluation metrics are:

Evaluation metric	recommendForSubset(500)	transform(test)
MAP	0.00882194609069326	0.43658624127599793
Precision At (500)	0.008015081967213115	0.1165455142951035
Ndcg At (500)	0.029310158250691002	0.5776484191370399

Selecting methods for prediction and evaluation

There are two methods used to generate recommendations:

- model.transform(test)
- model.recommendForUserSubset

According to documentation and our observation, model.transform(test) takes in the interaction matrix for the test set and predicts the user’s rating for each item occurring in the test set.

Model.recommendForUserSubset takes in a subset of userid, and returns the top k recommendations for that user. The recommendations are based on the order of calculated ratings from the latent factors. As a result, transform(test) will only recommend items already occurring in the test interaction set, which is the reason why we get better performance with the evaluation metrics. In practice, we should be using recommendForUserSubset to do our recommendation.

Evaluation criteria
In evaluation, we decided to use evaluation metrics (MAP, precision k, Ndcg At k) to demonstrate our effectiveness in recommendation. To get an unbiased result, we decided to exclude items that already had interactions in the training set when calculating true book set.

Dealing with rating
The Alternative Least Square algorithm utilized matrix factorization by decomposing the input (sparse matrix of rating) into latent user factors and latent item factors.

In model fitting, practically the user-item interaction with low rating (e.g. 1, 2) will still be taken as more significant rather than no interaction. We suspect that under this mechanism our system will be likely to recommend items with features that a particular user has no interest in. After discussing with others, we tried to bypass this by mapping rating onto [-2, -1, 0, 1, 2], taking rating = 3 as a neutral rating and rating lower than 3 as dislike. In model fitting, by setting the hyperparameter ***nonnegative*** to true, we don't take into consideration ratings lower than 3 (including rating = 0).

With hyperparameters rank=20, maxIter=10, regParam=0.1 and map rating [1,2,3,4,5] to [-2,-1,0,1,2], the model performance is:

Evaluation metric	recommendForSubset(500)	transform(test)
MAP	0.002236009028179178	0.4366335446231105
Precision At (500)	0.006577704918032788	0.11654551429510346
Ndcg At (500)	0.017642374150791083	0.57764841913704

The performance with recommendForSubset is worse than our baseline model, thus we decided to leave the rating as original in model tuning.

Tuning and Implications
During the tuning process, we fitted our model on 2% and 25% of the data with the hyperparameter grid focusing on tuning rank(# latent factors), regParam(regularization), and alpha(confidence on rating). *{rank: [10, 20, 50, 100, 150], regParam: [0.001, 0.005, 0.01, 0.05, 0.1, 0.5], alpha: [0.5, 1.0, 2.0]}.* And evaluated the model with MAP, Precision k, Ndcg At(k) separately on transformer(test) and recommendForSubset(test), we also journaled the run time. With the results on evaluation metrics and runtime (detailed records are provided in a separate file), the observations are as follows:

- With rank = 10, 20 the performance is relatively bad and doesn't change with different regParam and alpha value, which indicates that if the number of latent factors is smaller than 20, we can't learn effectively about the variance of rating.
- With rank = 50, 100, 150, we observe that model performance increase for regParam from 0.001 to 0.005 and start to depreciate with increasing regularization.
- In general, the performance of our model doesn't vary a lot with different alpha values.

- Evaluation metric results for transformer almost remain the same, while the prediction time for transformer is significantly lower than recommendForSubset.
- The performance improved with higher rank: the best MAP achieved by rank = 50 is 0.030791, rank =100 is 0.03951, rank = 150 is 0.110033, rank = 200 is 0.137102. The runtime for training model with rank = 100 is 27min47sec, for rank = 150: 55min48sec, for rank = 150: 57min52sec, for rank = 200: 125min37sec.

From the observations, we draw several inferences/conclusions:

1. The prediction results for transformer remains the same, which indicates that tuning hyperparameters has small effect on recommending items from the test item set. However, as the performance for recommendusersubset does vary across hyperparameter values, the tuning process is essential in achieving optimal performance for recommending from the whole item set.
2. Although performance improves with higher rank, we choose rank =150 because for rank = 200 the runtime nearly doubled but the performance only improved a little.

In conclusion, the optimal hyperparameter set we came up with is {regParam = 0.005, alpha = 1.0, rank =150}, with evaluation results on validation set as {MAP: 0.110033, precision:0.057023, Ndcg: 0.25943}, and test results as {MAP: 0.1084109666877974, precision At (500): 0.055457704918032785 ,ndcg At (500): 0.2564817472444875}. Comparing with the baseline model:

recommendForSubset(500)	Baseline model	Final ALS model
MAP	0.00882194609069326	0.1084109666877974
Precision At (500)	0.008015081967213115	0.055457704918032785
Ndcg At (500)	0.029310158250691002	0.2564817472444875

Our model after hyperparameter tuning has improved significantly from the baseline model.

Extension 1: Cold start

Building a model to map observable data to learned latent factor

We've approached this problem with a multi-output random forest algorithm, taking book features extracted from metadata as input space and learned latent factors as output space. The features include the average rating of the book, the country code and language code of the book, the authors (at most 2) and the most popular two shelves where the book is located. The detail input space and output space is as follow:

Input space X: {country_code, language_code, popular_shelves, authors}
f: Multi-output Random forest
Output y: {value on position of latent factor}

We train our model on the features from goodreads and item factors extracted from the previously trained ALS model. The general idea of the multi-output random forest algorithm is to fit random forest model 150 times for each book to get prediction of the 150 elements within the latent factor representation vector. To count into the correlation between the 150 elements within the vector, we put the latent factor element before the predicting element as features into the random forest model. To get the item recommended, we first predict the latent factor of the new item, append the latent factor to the item factor matrix, and get predicted ratings from multiplying with user-factors. We then proceed by recommend 500 items with highest ratings.

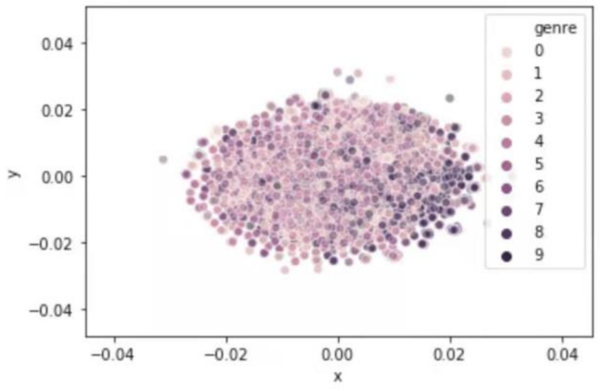
Results: The output of our algorithm does not seem promising. The MAP of the training dataset is 0.00896406 and the MAP of the test dataset is 0.00745872, which is lower than collaborative filtering algorithm. The possible reason might be that we did not put enough book information into consideration.

Extension 2: Visualization with T-SNE

T-SNE explained: T-SNE is a non-linear dimension reduction tool, it preserves the local structure of high-dimensional data by calculating scaled similarity score from the input space following a normal distribution and construct the output through first project the points stochastically onto the output dimensional space, then adjust the position by similarity scores following a t-distribution. In our attempt, we try to represent books with different genres using color from dense to light, taking item latent factors as input space and run T-SNE to plot on a 2-D space. Theoretically, if our model has learned about genres, then the latent factors in high dimensional space should form a structure with books clustered by genres.

Obtained results: In this case, we construct out input space with # of latent factors and run T-SNE with the following settings:

The resulted plot we obtain is as follow:



Observing from the plot, although the clusters are not clearly separated, we can roughly observe that dense colors are more concentrated at the bottom right part. It’s plausible that our model has embedded some genre pattern in the latent factors.

Team members and contributions

All the members participated in discussion of the project contributed suggestions during the process.

Chutang Luo: Running baseline model and tuning, extension 2.

Jiarui Tang: Extension 1.

Yutong Chen: Write-up and put together results.