DS-GA 1004 Big Data Final Project: Recommender System for GoodReads

Team members: Yue Ma (ym1970), Shuang Gao (sg5963)

1. Implementation

The first thing we did to preprocess the data was to convert the original csv file to parquet files, so we could save a lot of time accessing and analyzing the stored data. While preprocessing the data, we also delete users who have less than 10 records, because these users fail to provide enough information for evaluation. Next, we followed the requirements to split the data into training, validation and test datasets by creating a window partitioned by user ids of validation and test data frames, assigning row numbers to records in all partitions and transfering all validation or testing records with odd row numbers to train datasets. We then saved the resulting training, validation and testing datasets to parquets on HDFS for future parameter tunings. It is noteworthy that we have a parameter in our preprocessing function to specify the percentage of users we would like to sample before splitting the data. In this way, we could test our code on 1% of the data to make sure we wrote everything correctly and efficiently. The codes corresponding to this paragraph are included in *csv_to_parquet* and *preprocess* functions stored in *recommender.py* file.

For training, we used the alternating least squares (ALS) implemented by Apache Spark with the default maximum iteration, 10. For evaluation, we calculated 7 metrics for each combination of parameters: mean average precision (MAP), precision at top k and normalized discounted cumulative gain at top k, where k is set to 10, 200 and 500. There are a few parameters of spark configuration which, depending on the size of the data, should be adjusted for the program to run smoothly. As for the whole data set, we set the executor memory and driver memory to be 16GB and the number of executor cores to be 5. The codes corresponding to parameter tuning are included in *train, evaluation* and *parameter_tuning* functions stored in *recommender_submit.py* files.

Two group members contribute to this project. Yue Ma (ym1970) is responsible for building the pipeline to preprocess the data and train the model, figuring out reasonable spark configurations and tuning parameters. Shuang Gao (sg5963) is responsible for finishing the extension (exploration), tuning parameters and helping debugging. Both group members wrote the report.

2. Evaluation Results

As mentioned before, we evaluated our models on MAP, Precision at top k, NDCG at top k respectively. We set k to be 10, 200 and 500 to evaluate the model's performances when giving different numbers of recommendations. Although we calculated all 3 types of metrics, we decided to pay more attention to MAP and NDCG than to Precision, which does not take into account the order of the recommendations. For a uniform comparison, we considered MAP as the main metric to select the optimal model. Below is the results of parameter tuning on 1% sample.

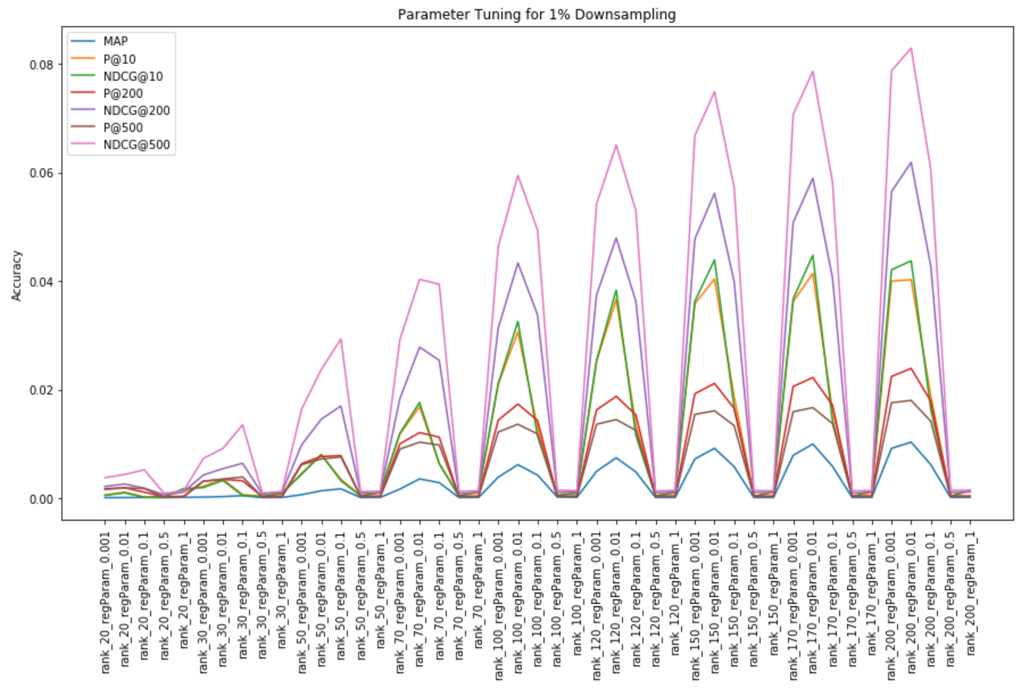

Fig 1. Evaluation results of different hyperparameter combinations in validation set of 1% samples.

As illustrated by the plot, the higher the rank is, the higher the accuracy is. This effect is especially significant as we increased the rank from 100 to 150 and is slowly flatten down afterwards. Also, given a rank, the accuracy reaches its peak when we set the regularization parameter to 0.01. These observations, though unable to support any conclusions on the whole dataset, provide us some guidance on the parameter selection when it takes a lot to train the full dataset on clusters. The chart below demonstrates the models we trained on the whole dataset. Same as the results on the 1% sample, higher ranks tend to provide better results. However, the regularization parameter behaves a little differently. RegParam(0.1) gives slightly higher MAP than RegParam(0.01). Based on the results on the validation dataset, we picked the model with 0.1 regularization and 150 rank as our optimal model. Using this model, we achieved the MAP of 0.000806 and the MDCG(500) of 0.016802 on the test dataset.

	MAP	P(10)	NDCG(10)	P(200)	NDCG(200)	P(500)	NDCG(500)
Rank(100) RegParam(0.01)	0.000221	0.000334	0.000324	0.000336	0.001442	0.000336	0.002370
Rank(120) RegParam(0.001)	0.000127	0.000211	0.000225	0.000178	0.000749	0.000164	0.001114
Rank(120) RegParam(0.01)	0.000413	0.000618	0.000613	0.000603	0.002586	0.000582	0.004206
Rank(120) RegParam(0.1)	0.000467	0.000730	0.000660	0.001838	0.004428	0.002487	0.010343
Rank(150) RegParam(0.01)	0.000797	0.001246	0.001197	0.001240	0.005086	0.001136	0.008012
Rank(150) RegParam(0.1)	0.000812	0.001298	0.001256	0.002051	0.007769	0.004011	0.016763

Table 1. Evaluation results of different hyperparameter combinations in validation set of full data.

3. Extension

From the 5 extension topics, we selected *Exploration* - to visualize the learned representations of users and items. In the models we trained and saved, for each user or each item, there is a representation vector, with the dimension the value of the rank parameter. By T-SNE, we can project these high-dimensional vectors to 2-dimensional vectors, and visualize each vector representing a user or an item as a point in 2-D space. We used the optimal model where rank is 150 and regularization parameter is 0.1 to visualize.

To illustrate how items are distributed in the learned space, we integrated information from the *Extracted fuzzy book genres*. There are 10 genre tags in the data: ‘children’, ‘comics, graphic’, ‘fantasy, paranormal’, ‘fiction’, ‘history, historical fiction, biography’, ‘mystery, thriller, crime’, ‘non-fiction’, ‘poetry’, ‘romance’, ‘young-adult’. We encoded them from 0 to 9 respectively. For each book, there is a ‘dictionary’ recording the value of times that each genre tag is extracted from users' popular shelves by a simple keyword matching process, and one book can have multiple tags. We selected the top genre tag as the ‘genre’ for each book, and saved this dataframe in *book_genre.parquet*. Since there are too many items to plot in the model with full data, we decided to visualize a part of them. We chose ‘children’, ‘history, historical fiction, biography’ and ‘mystery, thriller, crime’, which are encoded by 0, 4 and 5, and then sampled 50000 items from the three genres. As we expected, the map of the high-dimensional representations for different genres on the 2-D space demonstrated different distributions. The codes for the extension part are saved in *visualization.py* file.

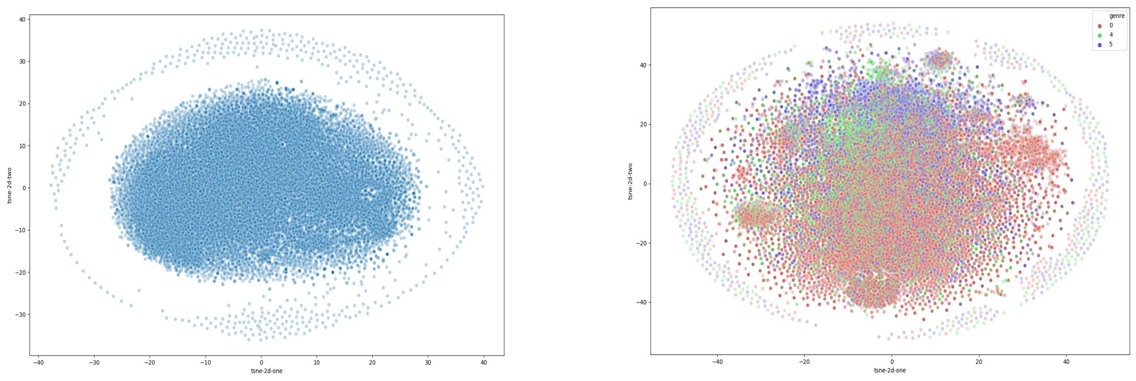

Fig 2: The scatter plot in the left shows user representations mapped to 2-D space. The scatter plot in the right shows item representations mapped to 2-D space, colored in genres.

In the second plot above, books of ‘children’ (red) present more at the bottom right of the map, while those of ‘history, historical fiction, biography’ (green) or ‘mystery, thriller, crime’ (blue) present mainly at the top of the map. Via observation, it seems like red clustering is more distant from blue and green ones, which is consistent with our intuition that books for children are more different from books whose readers are mainly adults. It is understandable that the three genres are not completely separable, because we use only fuzzy genres as additional information, and there are many other features influential on the representations.

Based on the plot, we find items sharing the same genre are located more closed in the learned space. So for a certain user, similar items can have similar predicted ratings calculated by latent factors, and thus the recommender system can recommend similar items according to the user’s previous preference.