MLT Final Project Report

Team: IWANTGPU B03902024 鄭筱樺 B04902036 梁友銓 R05725020 廖維君

1 Data Preprocessing

1.1 Reduce some noise

- ISBN: Remove invalid symbols: ('\',',','.','#','(',')','"','/',' ','-','+','*','"',':','=','>','<') and correct handwriting errors by replacing 'o' with '0'
- Author name: Remove space at the beginning/end of the string and invalid symbols

1.2 Generate features

- Age: Fill nan with mean or median
- Year-Of-Publication: Make invalid value (<0 or >2018) nan and fill nan with mean or median
- Author average rating: Average rating of all books written by a specific author
- Publisher average rating: Average rating of all books published by a specific author
- Author frequency: Number of books a specific author has written
- Publisher frequency: Number of books a specific publisher has published
- Title: Use pre-trained glove (100 dim) [1] to get the vector of title
- Description: Use pre-trained glove (100 dim) [1] to get the vector of description

1.3 Normalize features and target ratings

We perform standard normalization on all features and target ratings before training since normalization helps deep learning models when training since it makes each dimension has similar distribution.

2 Approaches

2.1 Matrix Factorization-based

2.1.1 MF

- Model description [2]
  - MF is a factorization of a matrix into a product of matrices. By decomposing rating matrix R (user_num x book_num) into U (user_num x hidden_dim) and B (hidden_dim x book_num), we can get the latent feature of books and users. We can predict unknown ratings by these latent features.
  - Formula: $\hat{r}_{ui} = \mu + b_\mu + b_i + q_i^T p_u$
    - μ: average rating in training data
    - b_u, b_i: user and book bias
    - p_u, q_i: user and book feature vector
- Implementation
  - With Keras [3] Embedding module, we map each user ID and book ID to a vector and then dot the two tensor vectors
  - We also add **bias term** since some user have preference to rate all book higher or lower than others

2.1.2 NMF

- Model description: MF with B and U to being **non-negative**.
- Implementation: Set embedding constraint to non-negative

2.1.3 SVD++

- Model description [4][5]
  - SVD++ can handle **implicit ratings**.
  - Formula: $\hat{r}_{ui} = \mu + b_\mu + b_i + q_i^T (p_u + |I_u|^{-\frac{1}{2}} \Sigma_{j \in I_u} y_j)$
    - Each book corresponds to two embeddings. p_u is for explicit rating, while y_j is for implicit rating.
    - When predicting a rating for (user u, book i), we consider all embeddings of I_u (books that user u has rated implicitly).
- Implementation

- Use implicit ratings in 'implicit_ratings.csv'
  - Add Embedding Layers for implicit ratings and sum implicit ratings of specific user u by creating a custom Layer.
- Use explicit ratings as implicit ratings: Use 'surprise' package [6]

2.1.4 MF+DNN

- Model description
  - Instead of dotting embedded user vector and book vector together, we **concatenate these two vectors** and feed it into a DNN. This allows machine to learn its own way to generate ratings from embedded user vector and book vector.
- Implementation
  - Substitute the dot part in MF with a Keras Concatenate Layer and add several Dense layers afterwards.

2.1.5 MF+KNN

- Model description
  - After MF, we get a full rating matrix. Replacing predicted rating with true rating in training data, we use KNN (k-nearest neighbor) to find user neighbors (users that have similar book preference) and book neighbors (books that get similar ratings). At last, we decide the final prediction by taking mean of those neighbor ratings.

2.2 DNN-based

2.2.1 Pre-trained Glove + DNN

- Features: Use user age, year of publication, author average rating, publisher average rating, author frequency, publisher frequency, title, description
  - Get title and description vectors by summing each word embedding
- Implementation: Directly use features as input and add multiple dense layers and dropout afterwards

2.2.2 Pre-trained Glove + RNN + DNN

- Features: Use user age, year of publication, author average rating, publisher average rating, author frequency, publisher frequency, title, description
  - Get title and description vectors by training a RNN
- Implementation: Concatenate RNN (for training title & description) with DNN

2.3 Tree-based

2.3.1 Random Forest

- Model description
  - An ensemble method.
  - Create many (e.g. 100) random sub-samples of the original dataset with replacement and then train a decision tree on each sample.
  - Combine predictions: Choose the class having the most votes (over all the trees in the forest) and in case of regression, it takes the average of outputs by different trees.
- Features: Use user age, year of publication, author average rating, publisher average rating, author frequency, publisher frequency, length of book description, length of book title.
- Package: sklearn [7]

2.3.2 Gradient Boosting Decision Tree

- Model description:
  - An additive regression model over an ensemble of trees, fitted to current residuals, gradients of the loss function, in a forward step-wise manner.
  - First fit a model to the data. Next, fit a new model to the residuals. Create a new model based on add the new model to the previous one.

- Features: Use user age, year of publication, author average rating, publisher average rating, author frequency, publisher frequency, length of book description, length of book title.
- Package: xgboost [8]

2.4 BS-ALS

- Model description [9]
  - BS-ALS is a baseline model that directly predicts the rating value by finding the best bias for user and book respectively with alternating least squares.
  - Formula: $\hat{r}_{\mathrm{ui}} = \mu + b_\mu + b_i$
- Package: surprise

2.5 Simple baselines

a. Average per user: Use the average rating of a specific user in training data. If the user has never rated any book, use overall rating average instead.
b. Average per book: Use the average rating of a specific book in training data. If the book has never been rated, use overall rating average instead.
c. Overall Average: Use overall average in training data.

3 Model comparison

	Track 1	Track 2	Popularity	Interpretability	Training time	# Parameters	Scalability
MF+DNN	**1.2132**	22.2855	High	Low	360 sec	4,533,270	Low
MF	1.2586	22.7204	High	Low	240 sec	4,484,226	Low
NMF	1.2239	22.6716	High	Low	240 sec	4,484,226	Low
SVD++	1.2497	22.6730	High	Low	600 sec	7,459,810	Low
DNN	1.4368	25.3138	High	High	60 sec	1,025	High
Tree (GBDT)	1.4341	24.9207	High	High	3.074sec (100 GBDT)	.	High
BS-ALS	1.2369	**22.2780**	High	Low	30 sec	263621	Low
Baseline	1.4331	25.2192	.	.	.		High

- We discover that MF-based models and BS-ALS far out-perform other models, which shows that **self-learned** user features and book features are important.
- Interpretability high: do not contain latent features that cannot be understood
  - DNN and trees are interpretable since they use real features, while MF-based models are not.
- Scalability high: do not need to re-train model if new users or new items are introduced
  - MF-based models need to retrain new user's or book's latent features, while DNN and trees do not since they use real features.

4 Ensemble

MAPE emphasizes on small value while MAE treat large or small values the same. Therefore, models that use MAPE loss have good predictions on small ratings. In this task, large values have more data. If we use MAPE loss, model will perform worse on large value prediction which reduce the accuracy a lot. As a result, we come up with an approach to deal with track 2: **If the prediction of model that use MAPE loss >= threshold, we give the prediction of model that use MAE loss more weight**. We use **grid search** to find the optimal weight and threshold on validation set.

Threshold	0 (all mae)	7	7.5	8	8.5	9	9.5	10 (all mape)
Valid MAPE	23.5555	23.1483	23.0204	22.9043	**22.8686**	22.9007	22.9181	22.9237
Test MAPE	.	.	.	.	22.5385	.	.	.

Weight: (more weight, less weight) = (1, 0)

5 Final recommendation for each track

5.1 Track 1: MF+DNN with loss MAE

- Public: 1.218708 / Private: 1.213378
- The biggest advantage is that this model only needs user rating as training data and can get quite a good prediction. Also its training time is quite short (It took 6 minutes to get best validation accuracy with one GPU in average). However, in real life, we will get more implicit ratings that explicit ratings, and those are data this model can't make use of.

5.2 Track 2: BS-ALS with loss MAE

- Public: 22.599608 / Private: 22.277990
- This is a very simple model but has good performance. The advantage is that it does not need any features of user or book and can be trained within a minute since number of parameters are very small compared to other models. Nevertheless, if there are useful features in the real world, the performance of this model can be defeated.

6 Experiment

6.1 MF+DNN

- Experiment settings
    - Hidden units in DNN layers: 128, 64, 32, 16; Batch size: 1024; Learning rate: 1e-4; Loss: MAE
- We've tried **different embedding dimension**. It shows that embedding dimension doesn't really matter. However, we choose 16 dimension as it takes least time to reach lowest validation accuracy.

Emb dim	Track 1 public	Track 1 private	Track 2 public	Track 2 private
16	**1.228408**	1.223887	**22.974582**	22.674718
32	1.232311	1.224981	22.981453	**22.665221**
64	1.229759	1.222771	22.999045	22.703472
128	1.228974	**1.222000**	22.993959	22.691487

- We've also tried something interesting: tried to scale the mean of final answer with different value (add a constant to the result to achieve this). Though we consider this approach as an overfit on testing set, the private score of "best overfitted model on public score" is surprisingly the highest score we have ever produced. We think the reason is that public test dataset and private test dataset have similar distribution.

Mean of final result	Track2 public score	Track2 private score
7.4	**22.532905**	**22.235499**
7.5	22.590895	22.294312
7.6	22.723920	22.430070
7.7	22.928301	22.637482

6.2 MF+KNN

- Since this model needs a lot of memory in order to count nearest neighbors, we reduce the popularity threshold (we only consider books that are rated by many users). However, this reduce the accuracy and leads to bad results.

6.3 SVD++

6.3.1 Use implicit ratings in 'implicit_ratings.csv'

- Experiment settings
  - lr=1e-4, Batch size = 512, Emb dim = 16, Loss = MAPE
- Best MAPE = 39.5124
- This does not work, so we switch to use explicit ratings as implicit ratings

6.3.2 Use explicit ratings as implicit ratings

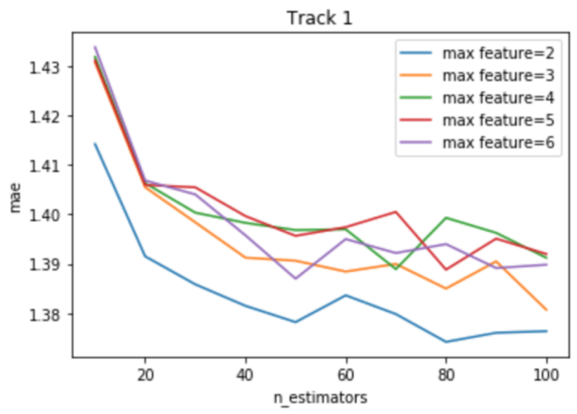
- Experiment settings
  - lr=0.007, Batch size = 512, Loss = MAE
- Surprisingly, embedding size = 1 performs best

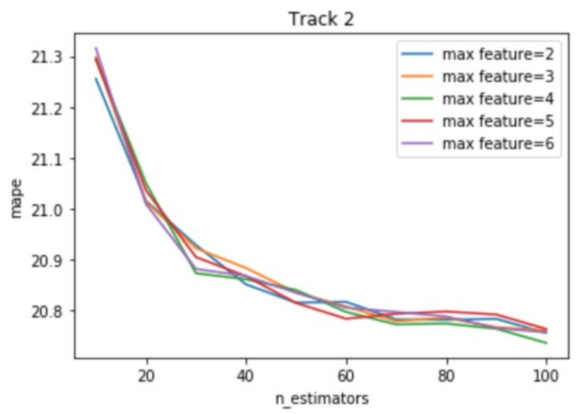
Embedding dimension	Track1 public score	Track1 private score	Track2 public score	Track2 private score
1	**1.257983**	**1.249695**	**22.989066**	**22.673014**
5	1.262671	1.253655	23.088431	22.770110
10	1.266182	1.257937	23.138202	22.798418
15	1.267094	1.258443	23.159200	22.837085

6.4 Tree-based

6.4.1 Random Forest

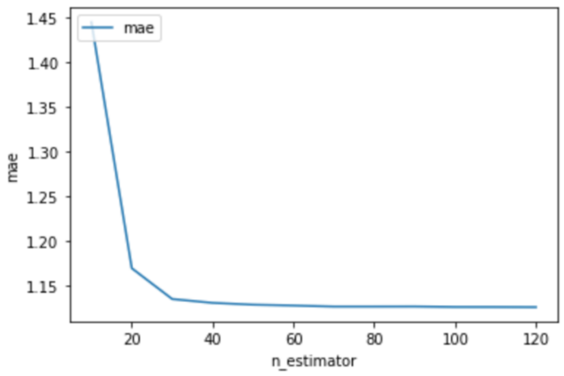
- We try different number of trees and number of features.

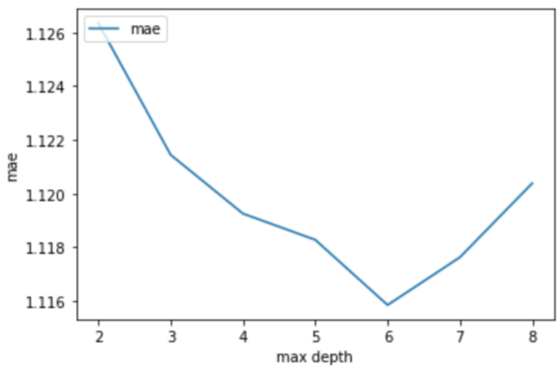




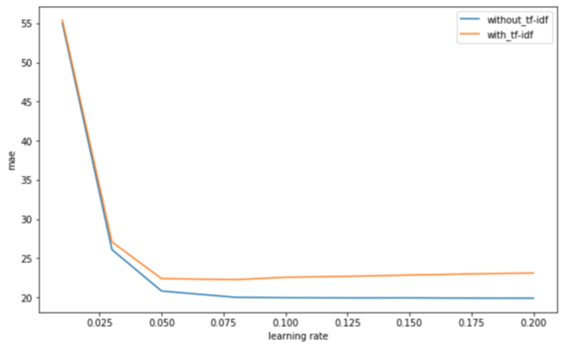
6.4.2 Gradient Boosting Decision Tree

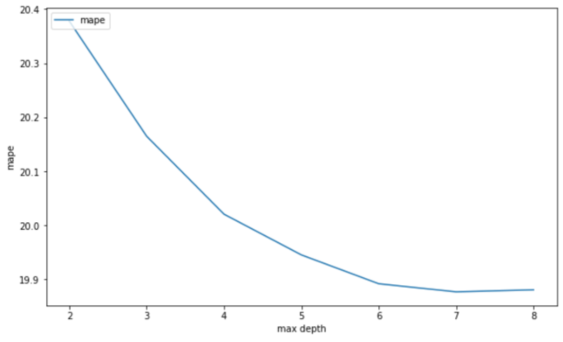
- Track 1
  - We try different number of trees and number of features.





- Track 2
  - We try different learning rate, different number of features.
  - Also try add tf-idf as features, and found it has no contribution in our experiment.





7 Work load

- B03902024 鄭筱樺: Data preprocessing, MF+KNN, (RNN+) DNN, Report
- B04902036 梁友銓: MF, NMF, MF+DNN, SVD++, Report
- R05725020 廖維君: RF, xgboost, simple baselines, Report

8 Reference

[1] Glove: https://nlp.stanford.edu/projects/glove/
[2] MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS
[3] keras: https://keras.io/
[4] Factor in the Neighbors: Scalable and Accurate Collaborative Filtering
[5] Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model
[6] Surprise: http://surprise.readthedocs.io/en/stable/index.html
[7] Sklearn: http://scikit-learn.org/stable/index.html
[8] xgboost: https://xgboost.readthedocs.io/en/latest/python/
[9] BS-ALS: http://surprise.readthedocs.io/en/stable/basic_algorithms.html