CSCI 567 Team 36 Final Project Report

Daoyuan Li
University of Southern California
daoyuanl@usc.edu

Yizhou Wang
University of Southern California
yizhouw@usc.edu

Yibin Xiong
University of Southern California
yibinxio@usc.edu

Abstract

In the final project, we explored enormous datasets of transaction history, customer demographics, and article descriptions from H&M Group to make personalized recommendations for the next 7 days after 2020/09/22. We learned about diverse models of recommendation systems with different assumptions and levels of complexity and extracted insights from the significant amount of data to produce good input features for models. Finally, we made an ensemble of models and achieve an MAP@12 score of 0.0239 on the leaderboard.

1 Problem introduction

In the era of big data, advanced algorithms are widely deployed in commercial contexts to model customer behaviors and make recommendations of products to different customers, which saves them from browsing through thousands of items online and enhances the shopping service. As one of the largest international fashion companies, H&M Group has collected approximately 2-year transaction data from 2018/01/01 to 2020/09/22 as well as the meta data of 1,371,980 customers and 476,930 articles for this Kaggle competition. Participants are encouraged to extract insights from the enormous amount of data and build a recommendation system that predicts the top 12 articles that each customer tended to buy in the next 7 days after 2020/09/22.

2 Data preprocessing and analysis

2.1 A first look and preprocessing

Firstly, we looked through each data frame to learn about what features we have and how many missing values are in each column. In `transactions_train.csv`, we have transaction records detailed with the date of purchase (`t_dat`), `customer_id`, `article_id`, `price`, and `sales_channel_id`, where `sales_channel_id == 1` indicates store sales and `==2` indicates online sales. There is no missing value in transaction data. In `articles.csv`, we are given hierarchical categories of the articles and text descriptions of some articles. Given the constraints on computational resources, we did not utilize the text descriptions and image embeddings. Additionally, we only used group number but not include group name to avoid multicollinearity. In `customers.csv`, we are given information about customers' habits on fashion news (`FN`, `fashion_new_frequency`), connection to H&M (`Active`, `club_member_status`), and demographics including `age` and `postal_code`.

missing values Note that there are small proportions (less than 3%) of *missing values* in `age`, `fashion_news_frequency`, and `club_member_status`, while there are over 60% missing entries

Preprint. Under review.

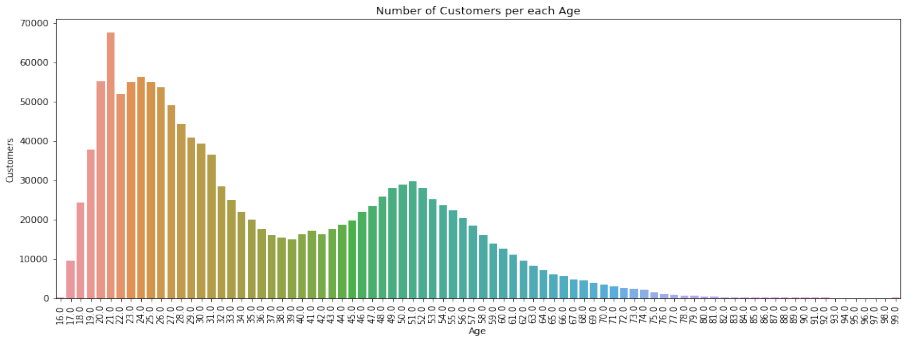

Figure 1: Customer age distribution

in `FN` and `Active`. For `age`, we simply fill the missing entries with -1 to indicate an anomaly. For the other features, we assumed that non-response is not likely to be random but reveals that the customer is apathetic to fashion news or the club, so we adopt *ordinal encoding* for these categorical features to indicate different active levels:

`fashion_news_frequency`: {`nan` or `None`: 0, Monthly: 1, Regularly: 2}
`club_member_status`: {`nan`:0, PRE-CREATE:1, ACTIVE:2, LEFT CLUB:-1}

and we fill the missing entries in `FN` and `Active` with 0's, as opposed to the 1's already there, to indicate indifference in fashion news or communication.

2.2 Hypotheses and data analysis

Time Although we have an extensive amount of transaction data, it does not mean that we can easily make effective use of all the data without considering time value. It is natural for customers to display *short-term* preferences over some items because of seasonal needs (e.g., Knitwear for autumn) or surging popularity of items recommended by influencers on social media. Therefore, we hypothesized that the closer to 2020/09/22 a transaction was conducted, the more likely that the purchased article was popular "recently" ("recently" means close to 2020/09/22). By analyzing Hyunjong's public notebook, we confirmed that there were relatively small changes of the trends between consecutive weeks, but sales could vary significantly in the long run. This inspires us to develop some baseline models based on time-decaying popularity and experiment with sequential models like GRU.

Re-purchase Another inquiry about customers' behaviors is whether people tend to re-purchase items that they are well accustomed to or try some fresh items that they have barely bought before. We follow Kaggle user LitchLab's procedures to compare the purchases in a given week with those in the previous 1, 2, or 3 weeks. It turns out that 10.8%, 14.3%, and 16.5% of customers have bought articles in the same index group as what they bought in the previous 1, 2, or 3 weeks, respectively. Given that ⅙ of customers consistently purchase similar items, it makes sense to make some "rough" recommendations just based on their consumption during 2020/09.

Customer age Typically, young adults constitute the majority of active users of e-commerce platforms. We explored the `age` data of all customers and plot the distribution (Figure 1). The distribution seems like a mixture of 2 Gaussian distributions with means at about 24 and 51. Therefore, we expect that people in their twenties have significant influence on the trending items and we should take the different preferences between age groups into account.

Customer gender Similar to age, gender can be a factor that exert remarkable influence on what kinds of articles that a customer is interested in. Exploring the sales of different index groups, we discovered that `Ladieswear` is the most frequently purchased one (Figure 2). With this result we infer that women customers can remarkably outnumber men or teenagers. Identifying this imbalance can help us distinguish consumers into different groups and make customized recommendations for each group.

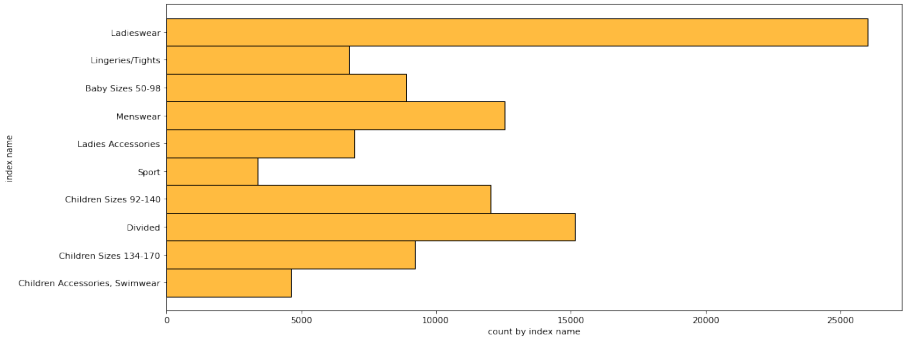

Figure 2: Sales counts of index groups

3 Models

3.1 Heuristic popularity index + clustering

3.1.1 Byfone's approach (fast trending)

Byfone's approach, based on trending and popularity, is one of the most common baseline models in this competition. This approach focuses on designing two new variables, `quotient` and `decay`. Specifically, `quotient` is defined as the quotient of the number of units sold in the latest week and that in the week of purchase for a given `article_id`. In other words, `quotient` represents the growth rate of each article from the week of purchase to the latest week. The other feature `decay` describes the number of days elapsed from the given week to the latest week, which represents the weight of each week. The final trending index for each article was generated accordingly by multiplying these two variables. Based on this statistical approach, the top 12 articles were selected and considered as the most popular ones. However, this implementation includes purchase history only in the past month instead of the whole timeline. In other words, we've assumed customers who have made purchases in the past month could represent the entire customers, so that popular articles are considered to apply to all customers.

3.1.2 Customers clustering

Given multiple features of customers, we've developed different strategies to classify customers. Since *age* has been found to be a decisive feature during prediction, we divided customers into several clusters statistically based on their age ranges. It would recommend the same set of articles to customers within the same age range based on the trending index in *Byfone's approach*. In addition to age, other features should also be taken into consideration. We've applied K-Means algorithm to divide all customers into 12 clusters, so that 12 sets of popular articles were used as the final prediction combined with *Byfone's approach*. In this case, customers in the same cluster were expected to receive the same recommendation.

3.2 Collaborative filtering: SVD

In recommendation system, one of the most fundamental yet effective approaches is *collaborative filtering*, which assumes that similar customers tend to prefer the same article (user-based) and similar articles are favored by the same person (item-based). This assumption enables us to leverage other customers' preferences to infer missing feedback from customer c_i to article a_j. Thus, the more customers we have in the training data, the more similar customers and items we can find for the target pair (c_i, a_j), which makes the prediction have less variance. This method seems suitable for our scenario of a huge dataset despite its prohibitive runtime.

One major class of collaborative filtering methods is matrix factorization. We can arrange the feedback into a matrix $X \in \mathbb{R}^{n \times m}$, where there are n customers and m articles. Usually X is a sparse matrix with a lot of missing entry for unobserved customer-article feedback. If A was fully observed, then we can perform singular value decomposition (SVD) to obtain $X = U\Sigma V^T$, where each column of $U \in \mathbb{R}^{n \times n}, V \in \mathbb{R}^{m \times m}$ corresponds to a customer or an article, respectively. By

keeping the first d dominant singular values, we can effectively embed each customer and article into a lower dimensional space $\mathbb{R}^d$, where each dimension may correspond to some crucial factor about customer-article interaction [Funk, 2013]. This means the reconstructions are close to the entries in X. Note that we can also factor X into two matrices instead of three by absorbing $\sqrt{\Sigma}$ into U, V.

If X is partially observed, then we only make the dot product of the corresponding vectors close to the observed entry through regression. Considering regularization, we want to minimize

$$\sum_{(i,j)\in\mathcal{K}} (X_{i,j} - u_i^T v_j)^2 + \lambda_u \|u_i\|^2 + \lambda_v \|v_j\|^2)$$

where $\mathcal{K}$ is the set of observed pairs and λ_u, λ_v are hyperparameters for regularization [Ma, 2008]. For this optimization problem, we can adopt typical machine learning approach by randomly initialize the matrices U, V and perform stochastic gradient descent [Ma, 2008].

Simon Funk has refined the SVD method further in the Netflix Prize competition. He introduced a *baseline* estimate μ that represents the "average" feedback among all customer-article pair and add the bias terms b_i, b_j in feedback prediction. b_i, for example, can capture the tendency of customer i to give better/worse feedback to most articles than an average customer. With baseline estimate and correction terms, $u_i^T v_j$ only needs to fit the residual and the loss function becomes [Koren, 2008]

$$\sum_{(i,j)\in\mathcal{K}} \left(X_{i,j} - (\mu + b_i + b_j + u_i^T v_j)\right)^2 + \lambda(b_i^2 + b_j^2 + \|u_i\|^2 + \|v_j\|^2)$$

In our implementation, we employed the GitHub repository reco. Based on the transaction records, we first produce a time-decaying "popularity index" by weighting each transaction by the inverse of number of days before 2020/09/22 and summing all the weighted transaction counts for each customer-article pair. This "popularity index" is used as feedback (or analogous to ratings in other cases) to feed into the FunkSVD model. Then we trained FunkSVD with stochastic gradient descent for 80 iterations.

3.3 GRU4RecF

Gated recurrent units (GRU) is a variant of RNN which has been extensively taken advantage in NLP areas because it resolves the vanishing gradient problem and requires less memory than LSTM. In our task, GRU4RecF was implemented to predict future articles based on previous articles and articles' features [Hidasi et al., 2016]. Item-to-item based GRU4RecF algorithm takes advantage of parallel architectures that utilize article representations for session modeling, which means GRU4RecF consists of multiple RNNs and each represents an article feature. The output states of all RNNs are merged to produce the score for all possible articles that a customer may purchase in the future.

Due to the limitation of available memory, we cannot input all data for training at the same time. Therefore, only transaction data after 2020 are selected, which based on the assumption of recent data can better represent users' current preferences. In addition, we divide users into three groups according to their purchase frequency. Thus, three GRU4RecF models are trained separately based on the three datasets. At last, the results of all models are combined and popular purchased articles have been selected to complement prediction results of unavailable customers.

3.4 LightGBM ranker

Light GBM is an efficient implementation of gradient boosting decision trees (GBDT), where a sequence of trees are deployed, and each decision tree fits the residual between the ground-truth and prediction in the previous decision tree. It is very economical in memory usage and supports parallelization [Ke et al., 2017], which suits our needs given the constraints on computational resources. In our problem context, we produce "positive examples" and "negative examples": a transaction record is "positive" if it indeed happens. Positive examples are simply extracted from the recent 10 weeks data (training data). For negative ones, we find out the top 12 bestsellers each week in the training data and concatenate them with all customers appeared to forge some transactions that did not happen. Whether a purchase happened is the variable to predict and we merge `customers.csv` and `articles.csv` to obtain all features we need for customers and articles. We also consider

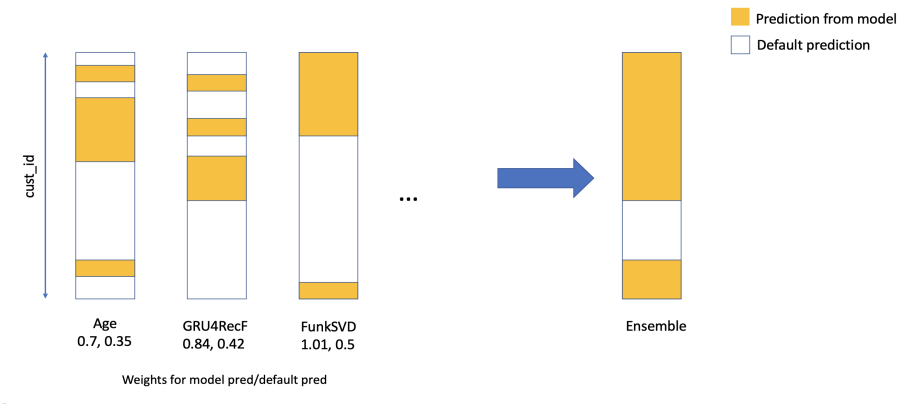

Figure 3: Two-phase dynamic weights

`bestseller_rank` (i.e., the ranking of a bestseller in that week, ranging from 1-12) as a useful feature.

With the training data, we applied the LGBMRanker model, which utilizes *LambdaRank* function to transform ranking into pairwise classification problem. Essentially, the model adjusts the ranking pairwise based on gradient signal with respect to the loss. This model achieves MAP@12 score 0.0211.

4 Ensemble

To ensemble our models mentioned in the previous context, we've decided to handle it creatively with two strategies.

4.1 Two-phase dynamic weights ensemble strategy

In this task, the customers of each model can be divided into two parts: those whose recommendations are be predicted by the model and those who are not in the training data and proposed with the generally popular items by default. Therefore, when we combine the results, it's unreasonable to assign the same weight for predictions from the model and default predictions, which are probably less accurate. For example, the overall performance of model A is better than model B because the number of customers predicted by model B is less than that of model A. However, the performance of the predicted part in B is better than A. If we only assign a higher weight to A, then the predicted part of B cannot get more attention as we supposed. In order to solve the problem, we introduced two-phase dynamic weights ensemble strategy. For each model, not only recommendation results of each customers will be given by the output, whether the customer has been predicted is documented as well. During ensemble process, higher weights will be given for predicted customers in each model compared to unavailable customers (Figure 3).

4.2 Adjustment of ranking weights

Considering each model makes a prediction of 12 articles for each customer, the order of these articles makes a great difference to the final prediction. Therefore, different weights are supposed to be assigned to each ranking properly, just as different weights between models, which is another dimension. Intuitively, the higher the ranking, the greater the weight. The key point is to find the balance between weights of models and weights of rankings. Our solution is to assign $\frac{1}{\sqrt{i}}$ as the weight for each rank i, in which case the difference of weights between high ranks are obvious while low ranks have close weights with each other. Combined with weights of models, we've got the final weight of each ranking of each model, which finishes our ensemble.

Table 1: Individual results

Models	Scores
Byfone's approach	0.0223
Customers clustering	0.0227
Collaborative filtering by SVD	0.0225
GRU4RecF	0.0221
LightGBM ranker	0.0211

Table 2: Ensemble results

Ensemble Strategies	Scores
Without any ensemble strategy	0.02329
Apply two-phase dynamic weights strategy	0.02384
Apply adjustment of ranking weights	0.02397
Apply both strategies	0.02399

5 Results

In this competition, we use Mean Average Precision @12 (MAP@12) as the evaluation metric.

$$\text{MAP@12} = \frac{1}{U}\sum_{u=1}^{U}\frac{1}{\min(m,12)}\sum_{k=1}^{\min(n,12)} P(k)\times \text{rel}(k)$$

where U is the number of customers, $P(k)$ is the precision at cutoff k, n is the number predictions per customer, m is the number of ground truth values per customer, and rel(k) is an indicator function equaling 1 if the item at rank k is a relevant (correct) label, zero otherwise.

The evaluation results of each individual models and ensemble strategy will be shown in Table 1 and Table 2.

6 How to run code

1. Separately, run `age.ipynb`, `byfone fast trending.ipynb`, `exponential-decay.ipynb`, `kmeans.ipynb`, `lgbm ranker.ipynb`, `svd.ipynb`, `time-is-our-best -friend-v2.ipynb`.

2. Run `GRU4RecF.ipynb` with index = 0, 1, 2, then run `merge_gru.py`.

3. Here are our prediction csv files for each model. At last, run h-m-ensembling-with-if-pred.ipynb.

References

Simon Funk. Netflix update: Try this at home, 2013. URL `https://sifter.org/~simon/journal/20061211.html`.

Chih-Chao Ma. A guide to singular value decomposition for collaborative filtering. *Computer (Long Beach, CA)*, 2008:1–14, 2008.

Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434, 2008.

Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 241–248, 2016.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.