HM Recommendation System

Author(s)
csci567_id05: Zhijie wu, Fanyou Meng, Honghu Luo

Abstract

In this project, we need to recommend top 12 personalized articles to each customer shown in the H&M dataset. In the process, we used some data preprocessing techniques to reduce the data size and applied several baseline models. Since the results of a single baseline model did not meet our expectations, we employed several ensemble techniques in the final step to further optimize the results.

1 Introduction

Team ID: csci567_id05

Team Member: Zhijie Wu, Fanyou Meng, Honghu Luo;

Team Strategy: Baseline + Ensemble.

1.1 Overview

To get our scheme closer to the given dataset, we first took a deep look at the importance of features (via sklearn.ensamble) and the data distribution of the dataset (provided by Kaggle). We then discovered that the dataset was so large that most matrix-based recommendation algorithms could not handle it. Therefore, we adopted various data preprocessing techniques in different baseline models to reduce the size of the H&M dataset, such as time constraints and clustering methods, etc. As mentioned in the abstract section, we experimented different baseline models, the best ones were: **SVD_Sort** (0.0221), **K-means + Predictions on transaction dates Strategy** (0.0226), **K-means + SVD_Sort** (0.0218). According to the characteristics and data requirements of different models, we performed different data processing and filtering techniques for each model, and fine-tuned the hyperparameters as much as possible. In the second section, we will introduce the specific details and corresponding codes of each baseline. After this, we chose to apply the ensemble techniques with the predictions generated by these better performing models, which will be described in detail in Section III. Meanwhile, in Section IV, we also briefly introduce other attempted schemes and their results.

2 Baseline

In this section, we will introduce some baseline models that we used and the results we obtained.

2.1 SVD_Sort Strategy

2.1.1 General Ideas

In this strategy, I employed a package called **reco [3]** , and we considered popularity to be one of the most valuable tags to help us make recommendations. Then, to combine popularity and purchase date, I added another column called pop_score, which means that if an item was purchased recently, it will have a higher pop_score than if it was purchased a long time ago; According to such tuple:

```
{'user':"customer_id", 'item':"article_id", 'value':'pop_score'}
```

I trained a singular value decomposition (SVD) matrix which was helpful for us to extract features and correlations from the user-item-pop_score matrix.

In the next step, I got four datasets with the highest volume of transactions and iterated through all the customers to find if they were in these four training sets, if so, this algorithm would work based on the results of the SVD and the list of most popular articles (via pop_score) to recommend. Otherwise, the algorithm would only make recommendations based on the list of most popular articles.

2.1.2 Data Processing and Cleaning

Data Processing: In this part, I paid more attention on recent data, so for the SVD matrix, I only selected 16 weeks of transaction data to build this model. I used pop_factor to show the popularity of one article, (which equals 1 divided by the number of days between the date of purchase and the last day of record) then summed up pop_factor of all articles as a new attribute named pop_score for each article.

Also, I took the transaction data of every four weeks as the four training sets, in which we could find the dataset of our customer group, and the smaller the sequence number of the dataset (from train1 to train4), the higher its priority.

2.1.3 Result

As a result, I ran this model for about 5 times with minor changes on parameters. For instance, I may revise the learning_rate of Funk_SVD to 0.002 and reset iterations to 150; then I increased the size of the training set. However, the results were not much different, the score always hovered around 0.0220, and the highest score via this algorithm was 0.0221 which was used as one of our baselines.

2.1.4 How to Run the Code

As you can see, there is a file in the zip file named H&M svd that is the code for this algorithm, and after run the script:

```
pip install -r requirements.txt
```

you can get all the dependencies needed, then revise the path of those source files like transaction.csv, article.csv and so on, to make sure it can read these files successfully, then run H-M-svd.ipynb file, you can finally get a submission.csv file.

2.2 K-means + SVD_Sort Strategy

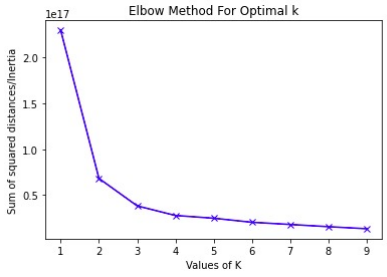
2.2.1 General Ideas

In this case, we decided to use K-means algorithm firstly to divide all customers to three clusters based on their attributes, and then divided transaction.csv to corresponding clusters. Then we used the separated dataset to train our SVD_Sort model (which has been introduced in the previous section) so that we could make more precise recommendation since different clusters might have different popular articles. The difference between this algorithm with the previous one is that we can find the most popular articles among a cluster of customers and make recommendation, which we thought was much more rational than the previous one.

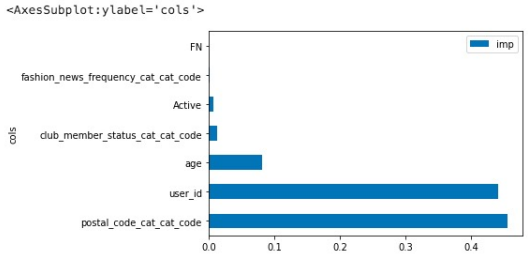
2.2.2 Data Processing and Cleaning

K-means: After trying several baseline models, we started thinking about clustering methods, which can be used to train individual recommenders for each group of customers and provide complementary information to the models. For clustering, we used the K-means algorithm which is supported by well-developed open source packages. The random forest regressor imported from the sklearn public

package can rank the importance of features from a dataset. It accepts the entire dataset as input and returns feature importance rankings. We decided to cluster based on customer information instead of article information, and let a random forest regressor rank the features of the customers. As the graph (b) shown, we used the top five user features as the input to the K-means algorithm. In K-means, it iterated from K= 1 to 10 and generated the graph (a), and finally we use the Elbow Method to determine the final value of K to be 3 (although 4 seems to be fine too). As the result, we clustered customers to 3 groups and produced a new file customers_clustered.csv.



(a) Elbow Method For Optimal K



(b) Feature Importance Ranking

SVD_Sort: After reading customers_clustered.csv, we divided the transaction data into 3 groups with respect to the corresponding customer cluster, and then used such (customer, transaction) pairs to make recommendations for customers in each group.

2.2.3 Result

SVD_Sort: Compared to the previous algorithm, this time I chose to use about 40 weeks' data to build SVD matrix since the data granularity of the previous version was not satisfied, and even revised volumes of the training set to cover more customers in our transaction data. (If we choose all transaction data, it will exceed the limit of RAM in the **google collab**). As a result, our final score was 0.0218. (If we didn't change the size of the training sets, the result didn't change much. Because when we only use 16 weeks' data for SVD, leading to lots of our prediction were just the same as the 12 most popular articles recently). Though our final score was not improved, we used this as another important baseline.

2.2.4 How to Run the Code

K-means: Simply run k-means-and-feature-importance-for-customers.ipynb with the raw H&M dataset as input.

SVD_Sort: After K-means, we can get a new csv file which has one more column named clusters to show which group this customer is in, then run h-m-Kmeans-svd.ipynb to run this algorithm, and finally will get a submission.csv file.

2.3 K-means + Predictions on transaction dates Strategy

2.3.1 General Ideas

This strategy uses the K-means algorithm to cluster customers into 12 categories with digital representation of all attributes of customers. Then we shaped the transaction into weekly purchases, and use the data in the last week to train our model. For customers that we don't have many data to analyze, we can use popular articles in the last week. Lastly, we use an equation with respect to x, which is number of day before the week that we are predicting on

$$(25000/\sqrt{x}) + 150000 * e^{-0.2x} - 1000$$

to compute the values of each transaction to make predictions. After combining processed prediction and popular articles, it can produce recommendations for each customer.

2.3.2 Data Processing and Cleaning

Processing: It changes all customer attributes from word representation to digit representation. And it uses the standardScaler normalization method to normalize customer data.

Cleaning: During data processing, it might involving many steps for copying data. Once we found out that it will not be used again, we will delete the data to prevent excessive RAM usage.

2.3.3 Result

This strategy makes predictions on each cluster group of customers at a time, and combines them at the end. It gives 0.0226 in comparison to our previous models, they are around the same. But this strategy is very time efficient and it only takes 150s to execute and that includes the reading process for huge .csv files.

2.3.4 How to Run the Code

After installing the dependencies and changing the path for input files, you can simply run the .ipynb file on kaggle with an accelerator GPU, and it will produce the recommendation for customers.

3 Ensemble

Generated better recommendation results by aggregating the results of each baseline model.

3.1 General Ideas

When our baseline models have reached a stalemate, we learned an ensemble approach from an open source material on Kaggle, which allowed us to reuse the predictions generated by the less-than-ideal baseline models and synthesized a more robust result - which demonstrated a much higher accuracy during the test.

3.2 Results and Methods

At very beginning, I selected the top 3 best performing models among all the baseline models: **SVD_Sort** (0.0221), **K-means + Predictions on transaction dates Strategy** (0.0226), **K-means + SVD_Sort** (0.0218). To reduce any possible bias, I first trained an **NCF** recommendation model with a small amount of data - this model computes each (customer, article) pair and then makes predictions. For this reason, it is difficult for this algorithm to make recommendations on such a large amount of data. However, here I limited its computations to the articles that have been recommended by the first three baseline models, which greatly reduced its workload and had the ability to re-examine all the previously recommended products from a more objective perspective. The final result was 0.0231 that was a huge improvement from the previous baseline models.

In the next step, we found an open source prediction which has a high test score (0.0238) and I applied the ensemble tricks again with it and all the predictions we generated before. The methods I tried was the following:

Majority Voting: This is a very simple algorithm that assumes all recommendation models have equal weights and selects the top 12 articles with the most mentions. The test result of this method finally reached 0.0239.

Average Weights: I used each model's test score divided by the sum of all test scores as the weight for each model and selected the top 12 highest weighted articles for each customer. The test result is 0.0239 that is indistinguishable from the majority voting method.

Random Weights Fine-tuned on the Dev Set: This time I used random weights to reduce the inherent bias that can be present with classical methods. The random weights were then slightly fine-tuned on a small development set. The final result reached 0.0240 after several adjustments.

3.3 How to Run the Code

Run recommenders-ncf.ipynb with one extra input res_blend.csv that aggregated the results from the top 3 baselines. The input file res_blend.csv is an output of h-m-ensemble-magic.ipynb. Run h-m-ensemble-magic.ipynb with input files sumbissions_{i}.csv, which are the outputs from baselines.

4 Conclusion

4.1 Challenge

Looking back on the entire project, we encountered many challenges such as large data volume and high time cost.

Large data volume: At the early stage of this project, we checked a lot of information online to give us inspiration to deal with this problem, and the first tough challenge we met was that the amount of our data was so huge that it always exceeded the limit of RAM in the **kaggle** platform, then we migrated our code to **Google Colab**. However, things didn't go well, then we had to choose partial data to run our code on some particular models.

High Time Cost: When we wanted to build a recommendation system, we needed to train corresponding model, which was really time consuming. For instance, when we used NCF to make predictions, it took more than 20 hours to train such a model, but it eventually reported errors because of the limit of RAM, which wasted a lot of time.

4.2 Achievement

Before the start of this project, the members of our group had not been exposed to such a large amount of data, so we did not conduct data screening but started building the model directly. However, due to the large amount of data, many models collapsed during the operation, and some of them were very expensive to generating predictions. In fact, we have tried more algorithms than mentioned in the report. Due to the difficulties, we had to re-observe the data and realized that data filtering can greatly improve the efficiency after reviewing various materials. For example, the clustering algorithm does a good job. Meanwhile, many features in a sparse dataset are probably not important at all, we found we can abandon them boldly.

In addition, through our actual practice, we observed that the amount of data used for model training is not the more the better. When we trained with the most recent 8 weeks of data, the results were not much different from 16 weeks of data, and the results of one year of data were more similar to the results of training with 16 weeks of data. What this actually tells us is that in such a dataset that is closely tied to the timeline, the distant data is likely to be meaningless, and this makes sense in reality. The type of product that customers loved 2 years ago may have fallen out of favor lately.

For this project, we slightly adopted customer features in training, and did not have a good method and not enough time to apply article features, which included important image information. We guess that's why our scores got stuck in a certain stalemate and struggled to improve. In the future, we will consider adding article and image information to the scheme design.

4.3 Failed Version

In this part, we will briefly describe some algorithms that have been tried but discarded due to unsatisfactory test scores.

4.3.1 Turicreate Model [1]

At the beginning, I thought this turicreate model was really helpful for our problem, and this article[1] was mainly talking about how to build a recommendation system based on **History Purchase Data**. Then I came to **Turi Api Documentation [2]** to learn about this model, and I found that this model had many strategies to make recommendation, for instance: Popular-based recommenders, item content recommenders and item similarity models (I mainly use these three strategies to build our recommendation system).

However, when I tried to use popular-based recommenders to plug in our training set, I found that our transaction.csv file was so large that it was impossible to use pandas.pivot_table to build a matrix for the whole data. (That always exceeded the limit of Ram in Kaggle Platform). Then I had to migrate my code to **Google Collab** and gave up normalized matrix. Then I tried to use collaborative filtering model to build this recommendation system via two different strategies: Cosine Similarity and Pearson Similarity to give recommendation based on user-item matrix, and even use some strategies like RMSE to evaluate our model. As a result, I took about one week to learn this model and built a recommendation system with three different strategies, but the highest score of all was only 0.00450. From partial to all data, I tested 7 times in total, each time combining different models and strategies to make recommendations, but didn't end up with a satisfactory score (much lower than expected). Therefore, I decided to give up this scheme and use other models as our baseline.

4.3.2 Two Towers

The Two Towers model is a deep neural network architecture that first analyzes the characteristics of customers and articles separately, and then aggregates the information together in the final layers to make recommendations. It was developed by tensorflow-recommenders and it already had a decent test score when I only entered 8 weeks of transaction data. However, after including all 16 weeks of data, this model took over 8 hours to train and 5 minutes per 1000 customers to make predictions. After finishing training the model, I didn't have enough time and equipment to support it to complete all the predictions, so I had to give up this method.

Reference

[1] Tjokro, Moorissa. "How to Build a Recommendation System for Purchase Data (Step-by-Step)." Medium, DataDrivenInvestor, 15 Oct. 2018, https://medium.datadriveninvestor.com/how-to-build-a-recommendation-system-for-purchase-data-step-by-step-d6d7a78800b6.

[2] "Recommender." Recommender - Turi Create API 6.4.1 Documentation, 9 Oct. 2020, https://apple.github.io/turicreate/docs/api/turicreate.toolkits.recommender.htm.

[3] mayukh18, Mayukh18. "Reco: A Simple Yet Versatile Recommendation Systems Library in Python." GitHub, 30 Mar. 2020, https://github.com/mayukh18/reco.

[4] "Introducing Tensorflow Recommenders." The TensorFlow Blog, 23 Sept. 2020, https://blog.tensorflow.org/2020/09/introducing-tensorflow-recommenders.html.

[5] Philipe, Hervind. "H&M: Faster Trending Products Weekly." Kaggle, Kaggle, 22 Mar. 2022, https://www.kaggle.com/code/hervind/h-m-faster-trending-products-weekly/notebook.

[6] Seeda, Pathairush. "A Complete Guide to Recommender System-Tutorial with Sklearn, Surprise, Keras, Recommender." Medium, Towards Data Science, 13 Oct. 2021, https://towardsdatascience.com/a-complete-guide-to-recommender-system-tutorial-with-sklearn-surprise-keras-recommender-5e52e8ceace1.

[7] Morty, Tarick. "[Lb 0.0240] H&M Ensemble Magic - Multi Blend." Kaggle, Kaggle, 29 Apr. 2022, https://www.kaggle.com/code/tarique7/lb-0-0240-h-m-ensemble-magic-multi-blend.

[8] Lebovitz, Ben. "K-Means and Feature Importance for Articles." Kaggle, Kaggle, 29 Apr. 2022, https://www.kaggle.com/code/beezus666/k-means-and-feature-importance-for-articles.