Harnessing Temporal Dynamics for Fashion Purchase Prediction: An RNN-based Approach

Roy Ludan 032736233 and Nadav Shaked 312494925

March 7, 2024

We train our models by using the following basic matrix: User-Based Collaborative Filtering (UCF), Item- Based Collaborative Filtering (ICF), Item Content-Based Filtering (ICBF), and Graph-based methods. the deep learning model we choose is Long short-term memory (LSTM).

1 Anchor

1.1 Setup Environments

For the execution of our project, we used a MacBook Pro with an Intel Core processor for tasks like exploring data, preparing data, and training traditional models. We set up our environment using Anaconda to make sure everything ran smoothly according to our project guidelines.
For deep learning models and ranker notebooks, we switched to Google Colab Pro. Here, we configured the runtime environment to use a T4 GPU for better computational performance. We also made sure to install all the necessary dependencies, including TensorFlow-GPU version 2.8.2, to match our project requirements.
By using these two environments, we could efficiently handle different aspects of our project and utilize the best computational resources for each task's needs.

1.2 Coding Effort

Our coding effort has been focused on executing all notebooks within our local environment. Initially, we faced challenges with installing TensorFlow version 2.8.2, which slowed our progress. However, after reinstalling necessary software and updates, such as Anaconda, we overcame this issue. We also addressed minor issues like updating file paths to ensure smooth functionality across all notebooks.
When attempting to train deep learning models locally, we encountered significant delays due to the lack of a GPU. We tried switching to another local computer with an NVIDIA GPU, but unfortunately, the CUDA version was incompatible with TensorFlow-GPU version 2.8.2, making installation impossible.
To overcome this obstacle and expedite training, we turned to Google Colab Pro, utilizing its T4 GPU capabilities. This decision yielded the expected performance improvements. We integrated previously generated data by uploading necessary files to Google Drive for easy accessibility within the Colab environment, ensuring continuity in our project execution.
After resolving all obstacles, we successfully executed all notebooks without encountering errors. Additionally, we split and edited notebooks to support various combinations of traditional algorithms with deep learning models for result exploration. This includes comparisons between UCF + LSTM, ICBF + LSTM, ICF + LSTM, Graph + LSTM, as well as running models for all traditional models, including LSTM, in the original notebook. This effort has helped us close any gaps that may have hindered our progress.

1.3 Execution Instructions

For instructions on running the notebooks, refer to our README.md file. This document contains detailed guidelines and steps to ensure seamless execution of the notebooks within our project environment.

1.4 Data Exploration

1.4.1 Session Statistics

We started by analyzing 1,000,000 browsing sessions. We found that the average session length was around 4.74, with a median length of 3.0. The standard deviation was approximately 6.08. These sessions included around 23,496 unique items, with the most frequently occurring item appearing 14,714 times. On average, each item was found around 201.89 times per session, the median was 95.0.

1.4.2 Purchase Statistics

Next, we dove into the purchase habits of users. The data indicates that items were purchased an average of 52.89 times, with the highest being 8,451 times, representing a wide range of consumer buying behaviors.

1.4.3 Item Feature's Statistics

Lastly, we looked at the specific features of items. The Dressipi dataset offered a broad range of 23,691 unique items, grouped into 73 different categories. These classifications contained 890 unique values for categories. Every item had a median of 20 features and an average of around 19.91 features. In total, the number of attributes varied from 2 to a maximum of 33 attributes per item. Notably, the same attributes (with same attribute ID and value) were found to have an average count of 48.51, with the maximum single-occurrence being 4,595.

1.4.4 Results

10-Fold Rankers					
	All traditional + LSTM	ICF + LSTM	ICBF + LSTM	ICF + LSTM	**Graph + LSTM**
MRR@100	0.25910	0.28780	0.25663	0.25663	**0.28816**
Normalized	0.18986	0.179329	0.15990	0.15990	**0.17955**
Training	91m	11m	20m	20m	**11m**

Table 1: Accuracy and runtime of the 10-Fold Rankers on the Train dataset

The results presented in the table indicate that the best performance, as measured by MRR@100 and Normalized metrics, is achieved by the Graph + LSTM model. Specifically, the MRR@100 for the Graph + LSTM model is 0.28816, which outperforms all other models listed. Notably, even when compared to the combined approach of LSTM with all traditional models, the Graph + LSTM model still exhibits superior performance. This suggests that incorporating graph-based information into the LSTM model yields significant improvements in recommendation accuracy. Interestingly, the training time for the Graph + LSTM model is also comparatively lower than other models, taking only 11 minutes. This finding raises the intriguing possibility that simpler models, such as the Graph + LSTM architecture, may offer a more efficient and effective solution for recommendation tasks. Thus, these results underscore the potential benefits of exploring less complex models in recommendation system design.

2 Innovation proposal

2.1 Challenges

Our main challenge with the innovation implementation part was understanding the different ranking systems we wanted to use along with the original *pm390/recsys2022* implementation. The original *pm390/recsys2022* implementation only used the LightGBM ranker as their prediction model output while we also used CatBoost and XGBoost in addition to the already implemented LightGBM. Our solution to this challenge was a vigorous study of the plethora of Ranking models used today by the industry, understanding their use, input, and hyper-parameter tuning. After a brief survey we concluded that selecting the aforementioned CatBoost, XGBoost and LightGBM as our ranking models

was the best choice for our use case. For example, we didn't use H2O as our ranking model due to it's sub par handling of categorical features.

2.2 Innovation implementation

we started by separating the different models of the original *pm390/recsys2022* implementation into different notebooks within the following categories:

1. XGBoost with Graph and LSTM
2. XGBoost with Traditional models and LSTM
3. CatBoost with Graph and LSTM
4. CatBoost with Traditional models and LSTM
5. LightGBM with Graph and LSTM
6. LightGBM with Traditional models and LSTM

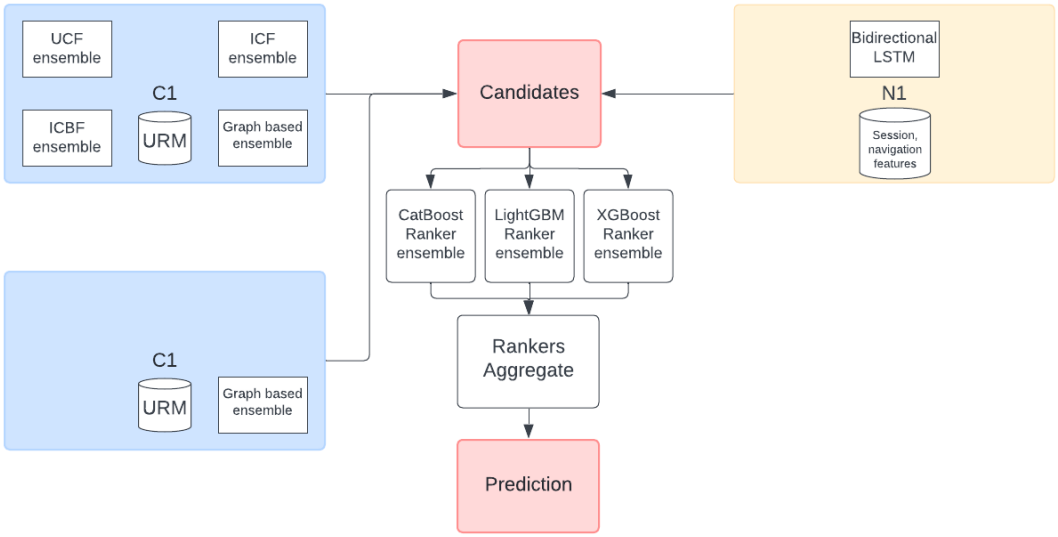

Figure 1: Ranker model architecture with the appropriate changes for the innovation proposal

Within each notebook we ran the training with the mentioned ranker model and candidate models. We ran all of the notebooks for the following datasets: Train, Leaderboard and the Final dataset. The output of every notebook we created was stored on a Google Drive.
The output of XGboost and CatBoost was a group of binary files and LightGBM outputted a text file. These files represent the architecture of the individual ranking models which were used as the inference (of ranking) for the various datasets.
We aggregated the ranks from the trained models that were generated from the Leaderboard and Final notebooks per each combination ranker model and candidate model. It is noteworthy that some of the combinations include more than one ranker model e.g. XGBoost with graph and LSTM along with CatBoost with Traditional models and LSTM.
The final result of the aggregate of each combination of ranker models was an add operation on the output of each ranker model. We then sorted the results according to their final score and finally returned the top 100 items per each session.
Some ranker models recommended items which were not found in another ranker model's results and this was solved by replacing the rank of the missing item with a value of 1000. This would prevent giving any weight to disagreeing ranker models as we are looking for a Consensus omnium (agreement of all).

2.3 Experiments

2.3.1 Candidate model selection and inference from the Test sessions

To predict outcomes for the test sessions, we retrain the Candidate Selectors using identical hyperparameters, incorporating data from all 17 training months. Subsequently, we utilize these updated models to generate candidates for sessions occurring in the 18th month. Leveraging the 10 previously trained LightGBM rankers, XGBoost rankers and CatBoost rankers, we forecast a score for each candidate. From these predictions, we select the top 100 candidates with the highest average scores for each session.

10-Fold Rankers					
	XGBoost + All Traditional & LSTM	XGBoost + Graph & LSTM	CatBoost + All Traditional & LSTM	CatBoost + Graph & LSTM	**LightGBM + Graph & LSTM**
MRR@100	0.25910	0.28780	0.25663	0.25663	**0.28816**
Normalized	0.18986	0.179329	0.15990	0.15990	**0.17955**
Training	91m	11m	20m	20m	**11m**

Table 2: Accuracy and runtime of the 10-Fold Rankers on the Train dataset for our innovation proposal

2.3.2 Ranking models results from the Leaderboard and Final datasets

We experimented with 2^5 model combinations in order to test if any combination and its aggregate would give an increased precision rate over the baseline of 0.1984 MRR@100 (The result of the *pm390/recsys2022* team from the leaderboard dataset).
We have divided the experimentation results into 3 categories: XGBoost, CatBoost, LightGBM. Each with two sub-categories: Graph & LSTM, All Traditional & LSTM. The value 'true' in the table indicates that the experiment included a model from the appropriate column.
For the sake of brevity, we only included the results from the Final dataset. The results for all datasets are included in the notebook supplied in the *4-dressipi-lgbmranker-innovation.ipynb* notebook.

XGBoost		CatBoost		LightGBM		MRR
Graph & LSTM	All traditional & LSTM	Graph & LSTM	All traditional & LSTM	Graph & LSTM	All traditional & LSTM	
true	false	false	false	false	false	0.1897
false	true	false	false	false	false	0.1926
false	false	false	false	true	false	0.1919
false	**false**	**false**	**false**	**false**	**true**	**0.1947**
false	false	true	false	false	false	0.1915
true	true	false	false	false	false	0.1908
true	false	false	false	true	false	0.1903
true	false	false	false	false	true	0.1916
true	false	true	false	false	false	0.1903
false	true	false	false	true	false	0.1930
false	true	false	false	false	true	0.1935
false	true	true	false	false	false	0.1930
false	false	false	false	true	true	0.1930
false	false	true	false	true	false	0.1919
false	false	true	false	false	true	0.1941
true	true	false	false	true	false	0.1922
true	true	false	false	false	true	0.1936
true	true	true	false	false	false	0.1917
true	false	false	false	true	true	0.1927
true	false	true	false	true	false	0.1915
true	false	true	false	false	true	0.1925
false	true	false	false	true	true	0.1942
false	true	true	false	true	false	0.1925
false	true	true	false	false	true	0.1942
false	false	true	false	true	true	0.1927
true	true	false	false	true	true	0.1928
true	true	true	false	true	false	0.1918
true	true	true	false	false	true	0.1926
true	false	true	false	true	true	0.1924
false	true	true	false	true	true	0.1939
true	true	true	false	true	true	0.1929
Mean						0.1925
Standard Deviation						0.0011
Median						0.1926

Table 3: Leaderboard dataset MRR results

2.4 Insights gained

1. We observed that while working with 3 different Gradient boosters, CatBoost was, in terms of training time, the slowest from the others in the pack.

2. Surveying the results of all the combinations we found that **None** of the combinations truly brought any meaningful improvement to the MRR@100 results.

3. We have learned that the rule 'simpler is better' also applies in this given case as the addition of more ranking models have made no real difference to the baseline of the *pm390/recsys2022* group's results.

4. As we can see from the stats of the MRR@100 results, most of the results were centered around 0.1925 with a STD of 0.0011 which means that none of the results, including the best one, have expressed a significant increase in accuracy.

5. In fact, some of the relatively better results came when we used only one Gradient Booster (LightGBM) and this is probably the case due to the *pm390/recsys2022* group's extensive study of its hyper-parameters which we used as is.

2.5 Further research

If we would have had the time, manpower and resources in the image of Graphics cards, Storage space, etc., we would have liked to look deeper into:

1. **Hyper-parameter tuning research:** We believe that further accuracy can be achieved with the current setup while utilizing the ranker models and the Gradient boosters's (XGBoost, CatBoost, LightGBM) hyper-parameters. This will require a much more in-depth expertise in operating these models although this could span for a very long period of time as the values set for the hyper-parameters can be very sparse.

2. **Aggregate ensemble ML model:** We would have created a ML model which would predict the final rank by giving it the output of all ranker models. We strongly believe that this model could provide a bird's-eye perspective into the various latent aspects of every ranker model which will give us a more precise result.

3. **Cross train the ranker models:** We would train the ranker models on similar data from a different and comparable source (another fashion website i.e. Asos, Next, Primark, etc.). This would emphasize the trend in certain item properties such as: color, seasonality, etc.