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

Roy Ludan 032736233 and Nadav Shaked 312494925

February 29, 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 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.

## 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.

## 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.

# 4 Data Exploration

## 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.

## 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.

## 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.

# 5 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** |

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.