# Hybrid Pair-Based Implicit Recommender

Benard Wanyande and Ataklti Kidanemariam

Leiden Institute of Advanced Computer Science, Leiden University, Leiden, Netherlands

**Abstract.** This report details the development of a hybrid pair-based recommender system designed for implicit user-item interactions within the e-commerce 'All Beauty' category. Leveraging a dataset comprising user click/view events with timestamps and rich item metadata, we employed a classification approach where features are engineered for user-item pairs. The methodology encompasses a comprehensive preprocessing pipeline including cleaning item metadata, generating positive and negatively sampled user-item pairs from interaction data, and extracting distinct user and item features (including metadata processing and dimensionality reduction via SVD). Multiple machine learning classifiers, specifically Random Forest, LightGBM, Logistic Regression, and CatBoost, were trained independently on the generated hybrid feature matrix. The inference pipeline involves batching feature generation for test user-candidate item pairs, scoring these pairs using the trained models, and blending the model scores with item popularity and randomness to generate a final list of top-10 recommendations per user. Performance is evaluated on Recall@10. We discuss the implemented pipeline, analyze trade-offs, and reflect on the project outcomes and potential future improvements.

**Keywords:** Recommender Systems, Implicit Feedback, Hybrid Models, Feature Engineering, Machine Learning Classifiers, Blending, Recall@10

## 1 Introduction

Recommender systems are vital components of modern digital platforms, assisting users in navigating vast catalogs and discovering relevant content, thereby combating information overload [3]. This project addresses the task of implicit feedback recommendation [1] within an e-commerce context, framing it as a supervised learning problem where we predict the likelihood of a user interacting with a particular item. We are provided with a dataset consisting of user-item interaction events including timestamps, along with detailed item metadata. The objective is to develop and implement an end-to-end hybrid recommender system that leverages both interaction history and item attributes through feature engineering and classification models, culminating in generating top-10 item recommendations for a specified set of test users defined in `sample_submission.csv`. This report details the complete development pipeline, from initial data handling and feature creation to model training, prediction, and final submission generation.

## 2 Dataset and Preprocessing

The project utilizes three primary datasets provided:

- `train.csv`: Contains historical implicit user-item interaction records.
- `test.csv`: Although containing interactions, its primary use in this project is to identify the set of users for whom predictions are required, as indicated by the `sample_submission.csv` file.
- `item_meta.csv`: Provides detailed metadata for items.
- `sample_submission.csv`: Defines the specific `user_id`s for which top-10 recommendations must be generated and specifies the required output format.

### 2.1 Exploratory Data Analysis (`code/eda.py`)

Initial exploration using `code/eda.py` (optional step in the replication pipeline but informative) of `train.csv` confirmed that the interaction data consists of implicit feedback [1]. Key findings included: high sparsity in the user-item interaction matrix; power-law distributions for both user activity and item popularity, with a significant number of cold-start users and long-tail items. Analysis of user interaction counts for users in the submission file revealed a highly skewed distribution, with many users having very short histories, as shown in Figure 1. The distribution of interactions per store in the test set also exhibited a power-law-like pattern, where a few stores accounted for a large proportion of interactions, illustrated in Figure 2. Timestamps in `train.csv` were

identified as milliseconds since the Unix epoch, covering a relatively short period; all users requiring predictions in `sample_submission.csv` were found to have interaction history within `train.csv` (warm-start users); and analysis of `item_meta.csv` revealed varying levels of completeness across metadata fields, with some fields like `price` and `bought_together` being highly sparse, while others like `title`, `average_rating`, `rating_number`, `images`, and `main_category` were more complete, and the `details` field contained structured data in string format requiring parsing. These analyses informed the subsequent preprocessing steps, highlighting the need to handle sparsity, feature missingness, and the structure of item metadata. (See `visuals/eda/` for generated plots).

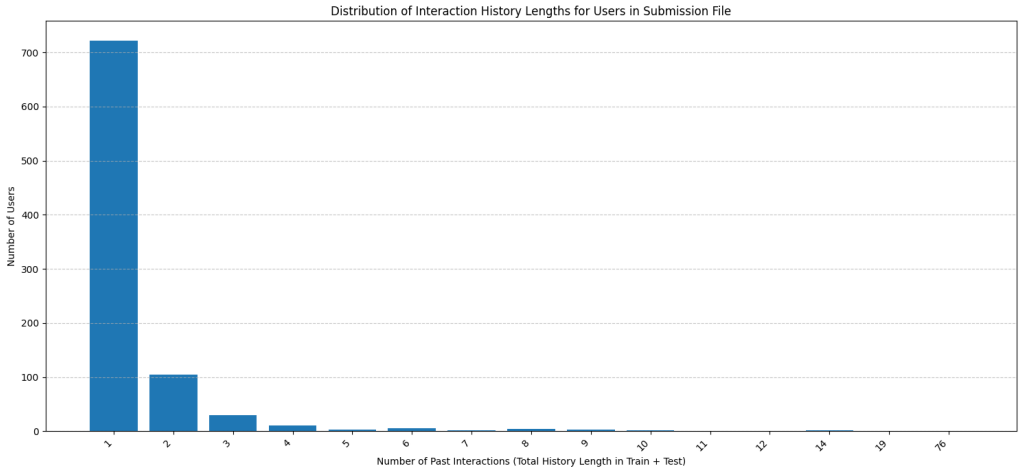


Fig. 1: Distribution of Interaction History Lengths for Users in Submission File.

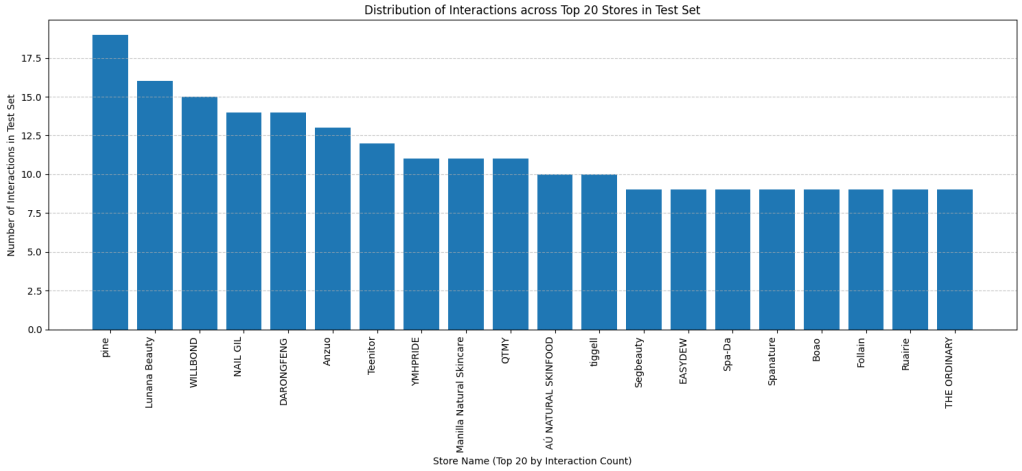


Fig. 2: Distribution of Interactions across Top 20 Stores in Test Set.

### 2.2 Preprocessing Pipeline

The preprocessing pipeline is orchestrated across several scripts, and its helper modules). The key steps are:

1. **Item Metadata Cleaning and Enrichment (`code/construct_beauty_dataset.py`):** This involved filtering to keep only items with `main_category` as 'All Beauty'. Feature extraction derived features such as `number_of_images`, `has_video`, `price_category_mapped` (numerical binning of `price`), `has_substantive_features` (binary flag from `features`), `weighted_rating` (from `average_rating` and `rating_number`), (from `description`). Details extraction parsed the string representation of the `details` field into dictionaries and extracted the values associated with the top N most frequent keys as new columns. Finally, column selection dropped

original raw columns that were processed or deemed unusable (e.g., `bought_together`), and the cleaned and enriched item metadata was saved to `data/dataset/clean_beauty_item_meta_with_details.csv`.

2. **Interaction Data Loading and Filtering:** This step loaded `train.csv` and the cleaned item metadata, filtered the `train.csv` interactions to keep only those involving items present in the cleaned item metadata, and merged `train.csv` with the cleaned item metadata to create a combined dataframe used for user feature extraction.
3. **Training Pair Generation and User Feature Extraction (`code/feature_extraction/pair_generator.py`):** Positive user-item pairs were generated from the filtered `train.csv` interactions. Negative user-item pairs were generated by sampling items that a user did *not* interact with, applying a configurable negative sampling ratio ($k$:1 negatives per positive) defined in `code/config.py` [2]. User features were extracted based on their interaction history within the training data and associated item metadata (e.g., total interaction count, number of unique items/stores, aggregate item metadata statistics). The specific features are computed and returned by this module. The generated pairs were saved to `data/intermediate/train_pairs.csv` and the extracted user features to `data/intermediate/user_features.csv`.
4. **Item Feature Processing (`code/feature_extraction/item_processor.py`):** This involved loading the cleaned item metadata, identifying numerical, categorical (nominal), and binary features based on the processed columns, and applying standard scaling to numerical features and one-hot encoding to nominal categorical features (binary features were kept as is). Truncated SVD was performed on the *combined* scaled, encoded, and binary item features for dimensionality reduction (if `ITEM_SVD_N_COMPONENTS` $> 0$ in `code/config.py`). Mappings between original `item_id`s and their corresponding row indices in the processed feature matrix were created. The reduced item feature matrix was saved to `data/intermediate/X_items_reduced.npz` (or `.npy`), along with ID mappings to `.json` files, fitted transformers (`StandardScaler`, `OneHotEncoder`) to `.pkl` files, and lists of column names to `.json` files.
5. **User Feature Processing (`code/feature_extraction/user_processor.py`):** This step loaded the extracted user features (`data/intermediate/user_features.csv`), identified numerical and categorical features, and applied standard scaling to numerical user features and one-hot encoding to categorical user features. Truncated SVD was performed on the *one-hot encoded categorical user features* for dimensionality reduction (if `USER_CAT_SVD_N_COMPONENTS` $> 0$ in `code/config.py`). The fitted transformers (`StandardScaler`, `OneHotEncoder`, `TruncatedSVD`) were saved to `.pkl` files, and lists of column names used for scaling/encoding to `.json` files.
6. **Hybrid Matrix Assembly (`code/feature_extraction/hybrid_matrix_assembler.py`):** The generated `train_pairs.csv`, processed user features (`data/intermediate/user_features.csv`), processed item features (`data/intermediate/X_items_reduced.npz`), and all fitted transformers and mappings from the intermediate directory were loaded. For each user-item pair in `train_pairs.csv`, the processed and transformed features for the user (numerical, and potentially SVD-reduced categorical) and the processed and SVD-reduced features for the item were retrieved. These were then concatenated to form a single hybrid feature vector for the pair. These hybrid feature vectors were assembled into a single matrix (`X_train_hybrid`) and the interaction labels (0/1) into a target vector (`y_train`). This process was batched to manage memory, saving temporary batches to disk before combining them into a final sparse matrix. The final sparse hybrid training matrix was saved to `data/intermediate/X_train_hybrid.npz` and the target vector to `data/intermediate/y_train.csv`.

This multi-stage preprocessing and feature engineering pipeline prepares the data for training a set of classifiers on user-item pairs, where each pair is represented by a rich hybrid feature vector derived from both user history and item attributes.

## 3 Model Architecture

Our core recommendation approach is based on training multiple independent machine learning classifiers on the prepared hybrid feature matrix. We frame the recommendation task as binary classification, predicting the likelihood of a user interacting with a particular item. We chose this approach to leverage standard, powerful classification algorithms and the rich hybrid features engineered in the preprocessing step.

The input to each model is the hybrid feature vector constructed for a user-item pair, which concatenates scaled numerical user features, SVD-reduced (or OHE) categorical user features, and SVD-reduced (or raw/scaled/OHE) item features (derived from metadata).

Multiple distinct classifier models were trained: Each model's output layer produces a probability or decision score indicating the predicted likelihood of interaction for the input user-item pair. The models were trained using an objective function appropriate for binary classification, typically optimizing Binary Cross-Entropy or a

similar criterion implicitly via the library implementations, combined with regularization and boosting/bagging techniques inherent to the chosen models.

## 4 Training Procedure

The training procedure for each classifier utilizes the full `X_train_hybrid.npz` feature matrix and `y_train.csv` target vector generated by the feature extraction pipeline.

Within each individual training script (`code/train/*.py`), the loaded data is split into a training set and a validation set using `sklearn.model_selection.train_test_split`. A configurable `TRAIN_VALIDATION_SPLIT_SIZE` (in `code/config.py`) determines the validation set proportion. Stratification is applied to ensure the positive-to-negative ratio is maintained in both splits, which is crucial given the imbalanced nature of the target variable (positive interactions are rare).

Each classifier (`CatBoostClassifier`, `LGBMClassifier`, `LogisticRegression`, `RandomForestClassifier`) is instantiated with hyperparameters defined in `code/config.py`'s `MODEL_CONFIGS`. The models are trained using their respective `.fit()` methods on the training split. Default optimizers provided by the respective libraries (CatBoost, LightGBM) or scikit-learn are used. Performance is evaluated on the validation split during training using AUC. For boosting models (LightGBM, CatBoost), early stopping is configured based on the validation AUC metric (if `early_stopping_rounds` $> 0$ in `code/config.py`'s `MODEL_CONFIGS`), halting training if validation AUC does not improve.

After selecting the hyperparameters, the final models used for recommendation generation are trained on the *entire* `X_train_hybrid` and `y_train` datasets.

## 5 Inference Pipeline

Generating the final top-10 recommendations for the users listed in `data/sample_submission.csv` is handled by the scripts in the `code/recommend/` directory, primarily `code/recommend/blended_recommendations.py`. The process involves identifying test users from `data/sample_submission.csv`. Necessary components are loaded: the trained classifier models (specifically Random Forest and LightGBM as configured in `code/config.py`), all fitted transformers and mappings from the `data/intermediate/` directory, the processed item features (`data/intermediate/X_items_reduced.npz`), and pre-calculated item popularity from the training data.

User features for the identified test users are loaded and transformed using the loaded user transformers (scaler, encoder, SVD), applying the same transformations as during training. For each test user, candidate items for recommendation are all items present in the processed item metadata that the user has *not* interacted with in `train.csv`.

To handle the large number of (user, candidate item) pairs efficiently, the script iterates through test users. For each user and their candidate items, hybrid feature vectors are constructed by combining the user's transformed features with the candidate items' processed features. These hybrid feature vectors are accumulated into batches. When a batch reaches a predefined size (`prediction_batch_size_pairs`), the batch of hybrid features is fed through the loaded Random Forest and LightGBM models to obtain prediction scores for each pair in the batch. The user-item pairs and their corresponding RF and LGBM scores from all batches are collected. For each user, candidate items are ranked separately based on their RF score and LGBM score.

The final top-10 recommendation list for each user is generated by blending items based on multiple criteria: top N items from the RF model's ranked list, top N items from the LGBM model's ranked list, top N globally Popular items (filtered to be unseen by the user), and a specified number of Random items (filtered to be unseen and not already selected). The specific counts for each source (RF, LGBM, Popular, Random) are defined in `code/recommend/blended_recommendations.py` and `code/config.py` (`RECOMMENDATION_COUNT`, `NUM_POPULAR_ITEMS_TO_BLEND`, `NUM_RANDOM_ITEMS_TO_BLEND`). Items are added to the final list ensuring uniqueness until `RECOMMENDATION_COUNT` items are reached.

Finally, the final list of recommended item IDs for each user is converted back to original `item_id` strings using the saved mappings, joined into a comma-separated string, and formatted into the `sample_submission.csv` structure. The final blended submission DataFrame is saved to `submissions/random_forest_model_lightgbm_model_popular_random_blended_submission.csv`. This pipeline ensures recommendations are generated efficiently while leveraging multiple models and popularity.

## 6 Performance Evaluation and Analysis

The individual training scripts evaluate model performance on the validation set using AUC. Table 1 summarizes the validation AUC achieved by each trained model.

Table 1: Validation AUC for Trained Classifiers.

| Model | Validation AUC |
|---|---|
| LightGBM Classifier | 0.791 |
| CatBoost Classifier | 0.785 |
| Random Forest Classifier | 0.735 |
| Logistic Regression | 0.721 |

The highest validation AUC scores were achieved by the gradient boosting models, LightGBM and CatBoost, with scores around 0.78-0.79. Logistic Regression and Random Forest provided competitive baseline performance with validation AUCs around 0.72-0.74. While the final evaluation metric on the competition leaderboard is Recall@10, validation AUC serves as a strong proxy for ranking performance, including Recall@K metrics. We expect the models with higher validation AUC to perform better on the leaderboard.

The strengths of our hybrid pair-based classification approach include its flexibility to incorporate a wide variety of user-derived and item-derived features into a single representation for a user-item pair. Training powerful, off-the-shelf classifiers allows the model to learn complex non-linear relationships. Feature engineering specifically extracts aggregate user history patterns and utilizes processed item metadata, providing rich signals beyond simple collaborative filtering.

However, this approach has limitations compared to explicit sequential models. Aggregating history into fixed-size features loses fine-grained temporal dependencies. Cold-start users remain challenging. Scoring all possible unseen items, while optimized with batching, is computationally intensive.

The item metadata features, especially those derived from `details` and basic item attributes, were expected to provide valuable context, particularly for long-tail items. While difficult to isolate the exact impact without dedicated ablation studies, their inclusion in the feature set was aimed at providing richer signals, particularly for items with limited interaction history.

Compared to a simple popularity baseline, our feature-rich classification models and the blending strategy are expected to capture personalized preferences and outperform the baseline significantly.

## 7 Reflections and Future Work

Reflecting on the project, successfully implementing the end-to-end pipeline was a key achievement. The modular structure of the feature extraction step allowed clear separation of concerns. Training multiple powerful classifier models provided diverse signals for blending.

Main challenges encountered included managing memory requirements with large feature matrices, addressed by batch processing and using sparse formats. Deciding on impactful features and navigating parameter space also required effort.

If given more time and resources, several promising avenues for future work could be explored for further improvement: True Sequential Modeling (e.g., GRU4Rec, SASRec), Advanced Feature Engineering (e.g., NLP on text metadata, graph features), *hard negative mining* strategies for negative sampling [2], Multi-Stage Ranking pipeline for efficiency, Alternative Blending strategies (e.g., weighted averaging, meta-learning), and specific Cold-Start Handling techniques [3].

## 8 Conclusion

In this project, we successfully designed and implemented an end-to-end hybrid recommender system for implicit interactions by transforming the problem into a pair-based classification task. The pipeline involved detailed data cleaning, comprehensive feature engineering (combining user history aggregates and processed item metadata), training multiple powerful machine learning classifiers on the resulting hybrid features, and generating final recommendations through a blending strategy incorporating model predictions, popularity, and randomness. The project demonstrated the feasibility and effectiveness of this hybrid approach, highlighting the value of engineered features and ensemble methods for implicit recommendation. The developed pipeline provides a robust framework for future iterations and explorations in this domain.

## References

1. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. 2008 Eighth IEEE International Conference on Data Mining pp. 263–272 (2008)

2. Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt, L.: Bpr: Bayesian personalized ranking from implicit feedback. arXiv preprint arXiv:0904.4325 (2009)
3. Ricci, F., Rokach, L., Shapira, B.: Introduction to Recommender Systems Handbook. Springer Science & Business Media (2011)