YZV 311E Data Mining Project Report: Improving Customer Experience with Predictive Analytics for Replenishment Recommendations

Hasan Taha Bağcı
Artificial Intelligence and Data Engineering
Istanbul Technical University
Istanbul, Turkey
bagcih21@itu.edu.tr

Selman Turan Toker
Artificial Intelligence and Data Engineering
Istanbul Technical University
Istanbul, Turkey
toker22@itu.edu.tr

***Abstract*—This project presents a predictive analytics approach for forecasting when customers will replenish consumable goods within the next four weeks. Leveraging historical transaction data from a global e-commerce platform, we train several models—including XGBoost, Random Forest, and an LSTM-based neural network—to predict whether a customer will repurchase an item in 0, 1, 2, 3, or 4 weeks. We detail our data preprocessing, feature engineering steps, and modeling choices. Experimental results show that XGBoost outperforms the other methods in terms of accuracy and balanced predictive performance, achieving an accuracy of 57.7%. The findings support the feasibility of integrating advanced predictive techniques in real-world replenishment recommendation systems, ultimately enhancing user experience and inventory management.**

***Index Terms*—Recommender Systems, E-Commerce, Predictive Analytics, XGBoost, Random Forest, LSTM, Consumption Forecasting**

I. Introduction

Ensuring timely replenishment of frequently purchased consumables (e.g., personal care products, pet food, and household essentials) is critical for e-commerce platforms seeking to enhance user satisfaction. The ability to predict the exact week of repurchase enables more targeted marketing campaigns, optimized inventory planning, and improved user loyalty through personalized recommendations [1], [2].

Despite the significant growth of e-commerce and the availability of large-scale transaction datasets, accurately forecasting a customer's next repurchase date remains challenging due to irregular purchasing behaviors, seasonality, and competing products. To address these challenges, we investigate a multi-class classification framework—mapping day differences to weekly buckets (0–4)—that captures short-term purchase intervals.

This project begins with a survey of relevant literature, highlighting various recommendation and predictive models. We then present a detailed methodology for data preprocessing and feature engineering, followed by model training and evaluation. Our experimental results demonstrate that tree-based ensemble models, particularly XGBoost, outperform other baselines, while an LSTM-based solution shows promise when finely tuned.

II. Literature Review

Recommender systems in e-commerce have evolved from simple collaborative filtering techniques to more advanced methods that incorporate temporal dynamics and recurrent neural networks [3], [4]. Traditional collaborative filtering methods rely on user-item interactions but often struggle with sparse datasets and do not explicitly model inter-purchase intervals [1]. Content-based approaches that leverage product and user attributes can offer personalized experiences but can be prone to limited novelty [2].

More recent research has focused on time-series models and neural networks to capture temporal patterns in consumption data [5], [6]. Recurrent Neural Networks (RNNs) and their variants, including LSTM and GRU, demonstrate strong performance for sequence modeling and have been applied successfully to forecast product demands [6]. However, these models can be data-hungry and computationally expensive, making tree-based methods such as XGBoost appealing due to faster training times and interpretability [7]. Random Forests are also widely used for classification tasks due to their ability to handle diverse feature sets and missing values, although they may underperform when classes are imbalanced or feature engineering is minimal [8].

Despite these advancements, there remains a gap in applying these methods to explicitly predict the exact week of repurchase. Our work addresses this by exploring both advanced ensemble methods (Random Forest, XGBoost) and a PyTorch-based LSTM approach, comparing their strengths and limitations on a large-scale transactional dataset.

III. Methodology

A. Dataset Description and Preprocessing

1) Dataset: The dataset for this study is sourced from a leading global e-commerce platform. This dataset comprises four CSV files:

- ***transactions.csv***: This file contains historical purchase data, with each row representing a single item purchased by a given customer. Key fields include:
  - `customer_id`: A unique integer identifier for each customer.
  - `product_id`: A unique integer identifier for each product purchased.
  - `purchase_date`: The date (*YYYY-MM-DD* format) when the product was purchased.
  - `quantity`: The number of units purchased in a single transaction line.
- ***product_catalog.csv***: A comprehensive list of products with attributes:
  - `product_id`: The same identifier linking to *transactions.csv*.
  - `manufacturer_id`: Indicates the product's manufacturer.
  - `attribute_1`, `attribute_2`, ..., `attribute_5`: Categorical attributes describing, for example, product size, packaging, or subcategory.
  - `categories`: A list-like string detailing product categories, often in a hierarchical manner.
- ***product_category_map.csv***: This file details hierarchical relationships among product categories:
  - `category_id`: Local category code.
  - `parent_category_id`: Reference to a higher-level grouping, enabling multi-level category insights (e.g., *Cleaning Supplies* as a parent for specific disinfectant products).
- ***test.csv***: A holdout file for final predictions, containing:
  - `id`: A row index for submission alignment.
  - `customer_id`, `product_id`: IDs for which a prediction is required.
  - `prediction`: An empty column in which our model's output (0–4) will be filled.

Overall, the raw transaction logs capture a wide array of different products, purchase frequencies, and seasonal behaviors. While the dataset is extensive and offers a rich source of information (e.g., purchase history for each user), it also presents several challenges, such as:

- **Sparsity:** Many customers purchase infrequently, or buy specific products only once.
- **Imbalance:** A large portion of rows indicate a long interval before repurchase, or no repurchase at all.
- **Incomplete Category Data:** Missing category and product attributes are common, requiring imputation strategies.

2) Merging and Cleaning: We merge *transactions.csv* with *product_catalog.csv* on the `product_id` field to integrate product-level attributes into each transaction record. Any missing category fields are either imputed with a placeholder label (e.g., `Unknown`) or replaced by the most common value found within that attribute if logically appropriate. Figure 1 illustrates the distribution of the final labeled target variable—`next_purchase_weeks`—after feature engineering.

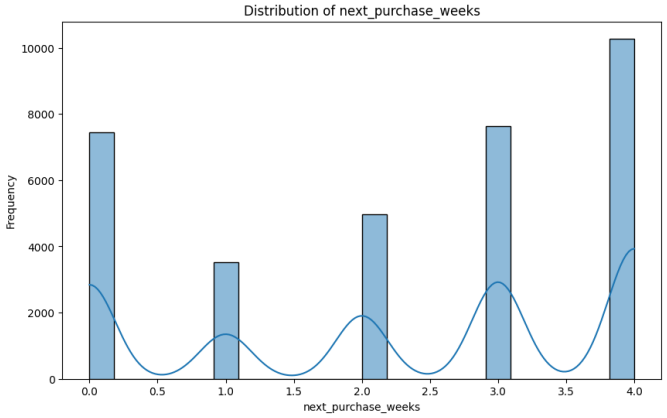

Fig. 1. Distribution of `next_purchase_weeks`.

B. Feature Engineering

1) Temporal Features:

- *Next Purchase Days/Weeks:* For each (`customer_id`, `product_id`) pair, we calculate the time difference between consecutive purchases. Specifically, `next_purchase_days` is computed as

  $$\text{next\_purchase\_date} - \text{current\_purchase\_date},$$

  giving us the number of days between two purchases. We transform these day-differences into weekly buckets, labeled from 0 to 4. Here, 0 corresponds to repurchasing in 4 or more weeks, and 1–3 correspond to repurchasing in under 4 weeks (1, 2, or 3). This encoding enables the system to handle short-term purchase behaviors and identify potential restocking patterns.
- *Recency:* We also include a `days_since_last_purchase` feature, capturing how recent the last purchase was. This allows the model to learn if a customer's repurchase cycle is getting shorter or longer over time.

2) Category Hierarchy Features: We utilize *product_category_map.csv* to link each product's category with its parent category. Since products can belong to multiple categories:

- *Parent Categories:* We create a `parent_categories` list for each product, where each item in the list denotes a parent category ID. This hierarchical context helps cluster similar items (e.g., cleaning supplies, personal care).
- *One-Hot Encoding:* We apply a MultiLabelBinarizer to transform each row's list of parent categories into a series of binary indicators. For instance, if a product belongs to categories 10 and 15, the transformed features will indicate `1` in the columns corresponding to these category IDs and `0` elsewhere. By embedding these encodings in the final feature vector, we allow the model to pick up on patterns across product groupings.

3) Derived Statistical Features: Beyond the basic merges, we incorporate additional derived features to capture customer-specific and product-specific nuances:

- *Customer Purchase Frequency:* For each `customer_id`, we compute the average time gap (in days) between consecutive purchases. This helps differentiate frequent purchasers from those with sporadic buying patterns.
- *Product-Level Attributes:* Attributes like `manufacturer_id` and other encoded `attribute_i` fields are included to incorporate possible brand loyalty or product type preferences. Missing values in these attributes are set to "Unknown" to avoid dropping rows.
- *Time-of-Year Effects:* While not extensively exploited in this study, we record the month of purchase to allow for potential seasonality detection. For instance, demand for certain household items may fluctuate depending on weather or holiday periods.

After these feature engineering steps, rows with no subsequent purchase receive a temporary label -1 and are dropped from further analysis. Additionally, to address the class imbalance where `0`-labeled transactions dominate the dataset, we sample 5% of `0` entries and combine them with all other labels, striking a balance between class representation and data availability.

C. Model Building

1) XGBoost: We train XGBoost (eXtreme Gradient Boosting) with a `multi:softmax` objective, 500 estimators, and a learning rate of 0.01. It efficiently handles feature interactions via boosting, often outperforming other models in structured data tasks [7].

2) Random Forest: As a baseline, we use a Random Forest with 500 estimators, a maximum depth of 3, and `max_features`=5. It aggregates multiple decision trees but may not always capture complex interactions if limited in depth [8].

3) LSTM Model: We also build a PyTorch LSTM with one hidden layer of size 64. The input dimension (581) includes product features, quantity, and the one-hot encodings of parent categories. Despite the sequence length being short (one row per transaction), we aim to capture potential sequential patterns.

IV. Experiments and Results

A. Experimental Setup

We split our final labeled dataset (`final_data.csv`) into train (90%) and test (10%). Hyperparameters for each model are tuned via grid search and early stopping (for XGBoost). After training, each model's performance is evaluated on the test set using accuracy, precision, recall, F1-score, and confusion matrices.

B. Performance Analysis

1) XGBoost Results:

- *Accuracy*: 57.7%
- Figure 2 shows the confusion matrix, indicating that class `4` (repurchase at $\geq$ `4` weeks) and `0` are well-captured, whereas classes `2` and `3` are harder to predict.

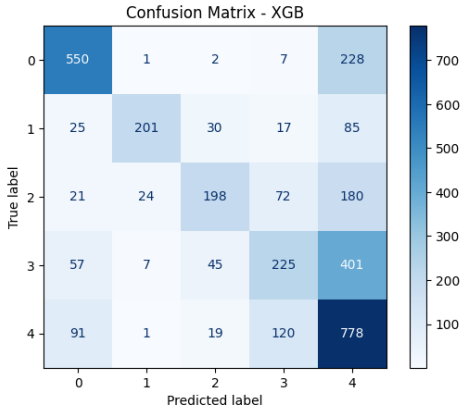

Fig. 2. Confusion Matrix - XGB.

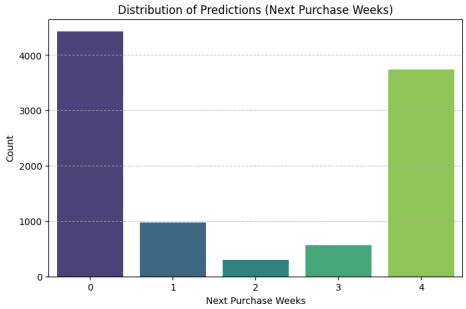

Fig. 3. Distribution of Predictions - XGB.

2) Random Forest Results:

- *Accuracy*: 29.8%
- The model heavily favors class `4`, leading to poor recall on other classes (Figure 4).

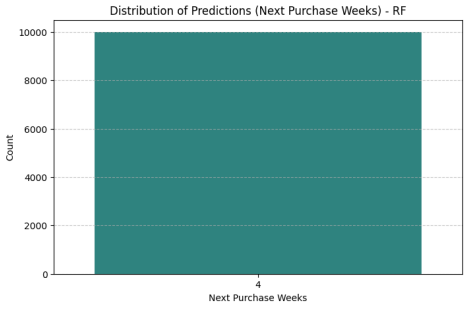

Fig. 4. Distribution of Predictions (Next Purchase Weeks) - RF.

3) LSTM Results:

- Converges slowly, predicting predominantly the dominant class 4, as shown in Figure 5.
- Requires more advanced sequence modeling and additional tuning to outperform tree-based methods.

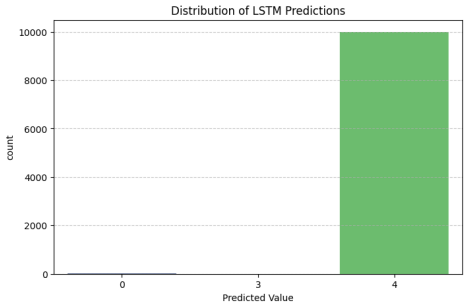

Fig. 5. Distribution of LSTM Predictions.

V. Conclusion

In this project, we proposed a multi-class classification strategy to forecast the specific week (0–4) in which customers repurchase consumable goods. After comprehensive data preprocessing—merging transaction logs with product attributes, engineering temporal features, handling category hierarchies, and sampling—we tested three predictive modeling approaches: XGBoost, Random Forest, and LSTM.

Our empirical findings demonstrate that XGBoost offers superior performance with an accuracy of 57.7%, indicating its efficacy in handling complex, structured data for short-term consumer demand forecasting. Random Forest struggled with imbalanced classes, while LSTM required further tuning and more advanced sequence modeling to realize its potential advantages.

Future work will explore more elaborate sequence definitions for the LSTM, additional features capturing seasonality, and possibly meta-ensemble strategies. As the e-commerce platform scales, these predictive models can be integrated into online recommendation systems to offer timely restock suggestions, improve inventory management, and ultimately elevate customer satisfaction.

References

[1] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Trans. on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[2] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in Artificial Intelligence*, vol. 2009, Art. no. 421425, 2009.

[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *IEEE Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[4] G. Zhang, M. Qi, and E. Patuwo, "Forecasting with artificial neural networks: The state of the art," *Int. J. of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[5] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. 30th Int. Conf. on Machine Learning (ICML)*, Atlanta, GA, USA, 2013, pp. 1310–1318.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, San Francisco, CA, USA, 2016, pp. 785–794.

[8] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.