Recommender Systems: MLDM Project

Zahir Ahmad [1] **Alejandro CARVAJAL** [1] **Amgad KHALIL** [1]

Abstract

This work addresses a data science challenge proposed by Carrefour through a Kaggle competition, focusing on predicting customer product repurchases in 2024 based on their 2022-2023 shopping histories. The dataset encompasses 87 million transactions from 100,000 customers, requiring substantial optimization that reduced the training data from 37.04 GB to 2.49 GB and improved load times from 450 to 3 seconds.

Exploratory analysis exposed key patterns in grocery e-commerce: transaction frequencies varying from single digits to over 1,600 per customer, basket sizes averaging 1.48 items, and mean reorder intervals of 12.76 days. Our methodological progression spanned statistical approaches to neural architectures. The customer-segmented statistical method achieved a Hit Rate @10 of 0.3605, surpassing simpler statistical models starting at 0.0760. Neural implementations showed lower performance, with NGCF reaching 0.124 on 1% of data and LightGCN achieving 0.15 on 10% of data. LightGBM, despite processing the full dataset, attained 0.17223.

The superior performance of statistical methods over neural architectures demonstrates that in grocery e-commerce, understanding domain-specific patterns and customer segmentation outweighs model complexity. These findings provide practical insights for developing large-scale recommendation systems in domains with irregular purchase patterns.

1. Introduction

This work addresses a data science challenge proposed by Carrefour through a Kaggle competition running from September 2024 to January 2025, with the primary goal of predicting product repurchases on their e-commerce platform. The challenge stems from the complex nature of grocery e-commerce, where purchasing patterns exhibit distinct irregularities - from weekly yogurt purchases to yearly pepper mill replacements - making traditional recommendation approaches less effective (Scherpinski & Lessmann, 2021).

The competition framework utilizes purchase histories from 100,000 customers during 2022-2023 to predict their first transactions in 2024. The data encompasses 87,037,462 records requiring significant computational optimization. Through careful data type restructuring and processing improvements, we reduced the training dataset from 37.04 GB to 2.49 GB while improving load times from 450 to 3 seconds, enabling rapid experimentation with multiple methodological approaches.

Our methodological progression incorporates both statistical and deep learning frameworks. The statistical approaches range from basic popularity-based models to sophisticated customer-segmented strategies (Koren et al., 2021). For deep learning implementations, we explored Neural Graph Collaborative Filtering (Wang et al., 2019) and LightGCN (He et al., 2020), examining their efficacy in capturing complex purchasing patterns. The LightGBM implementation provided insights into gradient boosting approaches for large-scale recommendation systems.

Performance evaluation centers on Hit Rate @10, measuring the percentage of actual purchases present within the top ten recommended products. This metric aligns with practical e-commerce requirements where customers typically view a limited number of recommendations. The evaluation framework splits customers into training (80,000) and test (20,000) sets for 2024 predictions, enabling robust assessment of model generalization.

This research contributes to the field through three primary aspects: demonstration of effective data optimization techniques for large-scale e-commerce data, comparative analysis of statistical and neural approaches in grocery recommendation systems, and insights into the relationship between model complexity and recommendation accuracy in domains with irregular purchase patterns.

2. Problem Statement

What sets the grocery e-commerce realm apart is its distinct and often irregular purchasing rhythms, which make the development of an accurate recommendation system particularly complex. For instance, whereas yogurt packs might

be purchased every week, other products (like pepper mills) might be bought only once in several years.

The temporal structure of the data is shown in Table 1. This table seeks to show how information collected from 100,000 customers for purchase histories in the year 2022-2023 may predict when such customers will first transact in the year 2024. With that, the system is expected to make recommendations of up to 10 products that each customer is expected to repurchase.

Table 1. Temporal Structure of Customer Data

Time Period	Customer Count	Purpose
2022-2023	100,000	Training
2024 (Known)	80,000	Test and Validation
2024 (Hidden)	20,000	Test

The performance metric used for benchmarking is Hit Rate in 10, which measures the percentage of items within the customer's actual first purchase in 2024 that was available in the top ten recommended products.

Formally, the Hit Rate @10 is given as:

$$\text{HitRate@10}(a, y) = \frac{1}{\min(N, 10)} \sum_{i=1}^{\min(N,10)} \mathbb{1}_{y_i \in \{a_1, \ldots, a_N\}},$$

where:

- $(a_i)_{1 \leq i \leq N}$, with $N \in \mathbb{N}^*$, are the products a customer actually purchases in their order;
- $(y_i)_{1 \leq i \leq 10}$ are the 10 products predicted for that customer.

Through this framework, our main objectives are:

- Predict the first 2024 transaction for 20,000 customers based on their purchases in 2022–2023,
- Recommend the top 10 products believed to be most likely to appear in their first 2024 order,
- Optimize the Hit Rate @10.

3. Exploratory Data Analysis (EDA)

Our initial exploration highlighted significant differences in data scale, which influenced both memory usage and modeling complexity. Table 2 provides an overview of record counts, columns, and file sizes.

Table 2. Dataset Overview and Memory Footprint

Dataset	Records	Columns	Size (GB)
Training Data	87,037,462	10	37.04
Test Data	1,220,706	3	0.23
Products Data	82,966	49	0.08

A notable observation was the different levels of missing values. The transactional data proved nearly complete, but the product attribute file contained various missing fields (see Table 3). This gap distribution guides how we may incorporate or impute these attributes in our recommendation engine.

Table 3. Missing Values in Product Attributes

Attribute	Missing Count	Missing (%)
brand_key	63	0.08
shelf_level3	1,313	1.58
shelf_level4	42,042	50.67
ecoscore	70,464	84.93

Given the completeness of purchase logs and the partial gaps in product attributes, a layered approach emerges: features readily available through transaction logs, complemented by product-level features (if not missing).

3.1. Customer Behavior Analysis

We studied transaction behaviors of 945,135 unique customers. The average was 92.09 transactions per customer, but the median was only 61, reflecting a right-skewed distribution. Indeed, some individuals had as many as 1,609 transactions, while others had far fewer. Figure 1 illustrates this disparity.

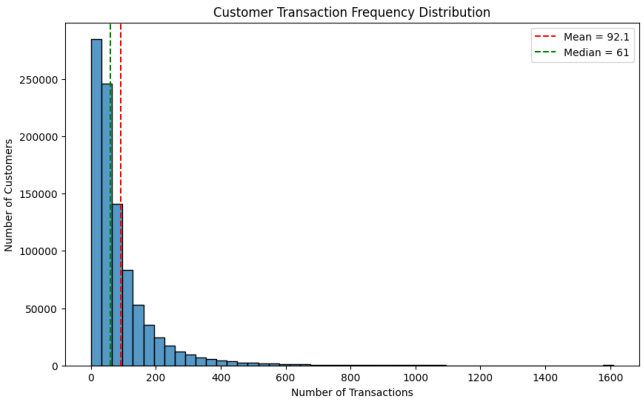

Figure 1. Customers' Transaction Frequencies

We also noticed that most orders contain few items: the mean is 1.48 items per transaction (std 0.47). Nearly 75% of purchases feature fewer than 1.59 items. The bar chart in

Figure 2 provides a succinct overview of that tendency.

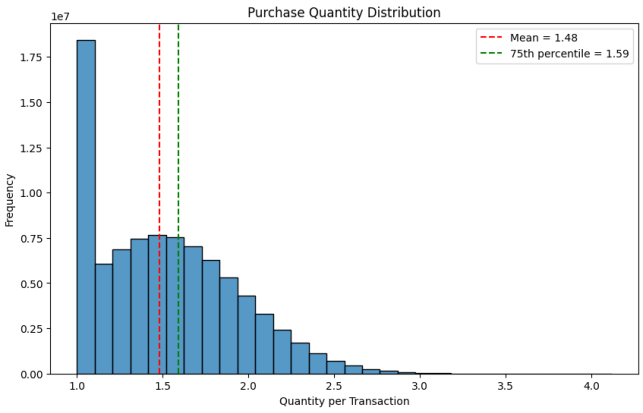

Figure 2. Purchase Quantity

3.2. Product Analysis

Our data suggests that just five departments dominate the bulk of all sales.

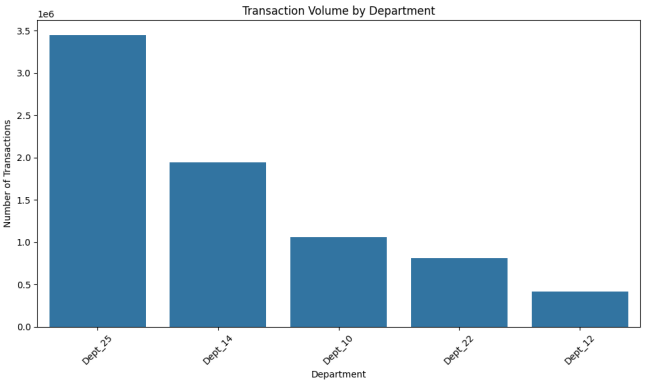

Figure 3. Transaction Volume by Department

Figure 3 encapsulates how, while there is breadth in product offerings, the transaction density is highly concentrated in certain categories.

In addition, we identified strong consumer focus on healthier, more environmental-friendly products, such as those labeled *palm oil free* or *pesticide-free*.

3.3. Temporal Pattern Analysis

Temporal patterns reveal cyclical repurchases: the mean reorder interval sits around 12.76 days, though it spans anywhere from days to hundreds of days (see Figure 4). Seasonal dynamics were also evident, with peaks in March (7.9M transactions) and relatively high activity in Q4.

In summary, EDA suggests focusing on 1. tailoring recommendations to different buyer archetypes (given widely varying frequency), 2. leveraging department-level focus, and 3. incorporating time-based signals. Our baseline Hit Rate @10 sits at 33%, and these findings will serve as guideposts for boosting performance.

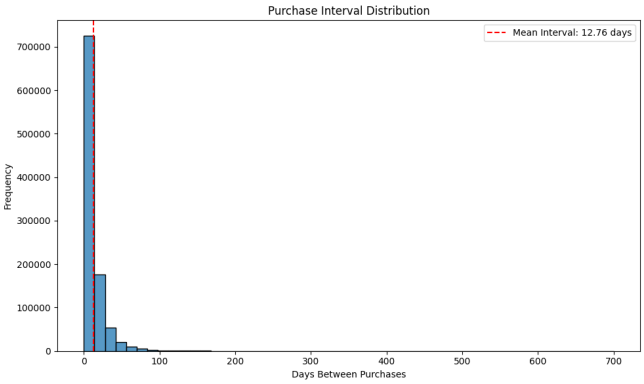

Figure 4. Purchase Intervals

4. Data Processing

Because the training data is so large, we applied careful optimization to reduce memory overhead and improve loading times. Initially, the dataset demanded around 37.04 GB of memory and took over 450 seconds to load.

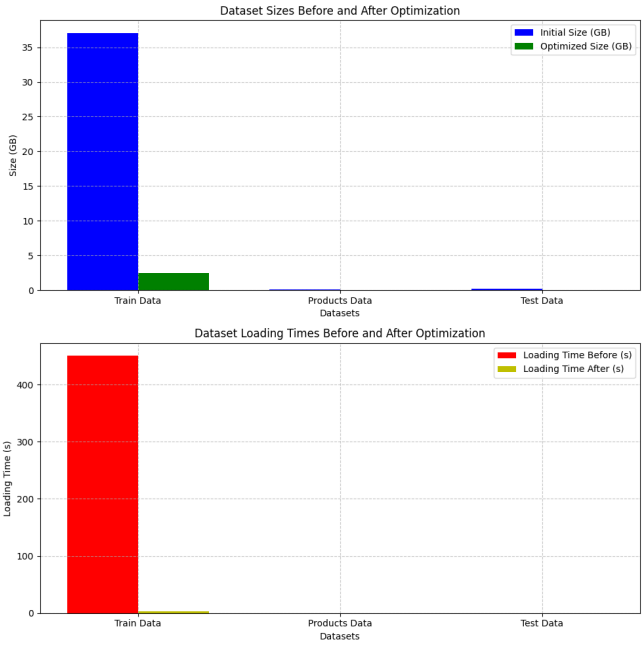

Figure 5. Data Optimization Effects

Through scrutinizing value ranges and assigning narrower data types (e.g., 8-bit for binary flags, 16- or 32-bit for columns with progressively larger ranges), we reduced the training dataset to 2.49 GB, an overall shrinkage of 93%. Similar transformations lowered the products file from 81.94 MB to 2.24 MB and the test file from 228.88 MB to 13.97

MB.

Simultaneously, data load times for training reduced from approximately 450 seconds to below 3 seconds, thereby significantly speeding up our experimentation cycles. We also learned that this type of holistic viewpoint of memory compression costs versus runtime costs of type casting was the crucial missing element for us. In total, these advances allow rapid iteration on model building without compromising the integrity of the data or important predictive factors.

5. Statistical Approaches

We tried out various methods, each based on patterns we noticed in the data. Our final Hit Rate @10 scores, which show how well our methods worked, went from 0.0760 to 0.3605. This improvement came from using more detailed and sophisticated techniques over time.

5.1. Popular Products (Hit Rate @10: 0.0760)

A straightforward baseline ranks products by total transaction volume:

$$\text{score}(p) = \sum_{t \in T} \mathbb{1}[\text{product}_t = p],$$

where T is the entire transaction list. Although certain departments (`Department_25`, `Department_14`, etc.) heavily dominated purchase volume, this method yielded only 0.0760, indicating that popularity alone is insufficient.

5.2. Recent Purchase History (Hit Rate @10: 0.1951)

Here, we inject temporal recency. For a customer c, we define:

$$\text{score}(p, c) = \sum_{t \in T_c} \mathbb{1}[\text{product}_t = p] \cdot \text{rank}(t)^{-1},$$

where T_c encompasses all transactions from customer c, and $\text{rank}(t)$ measures how recent a transaction is. Because frequent repeat purchases emerged as a strong signal, performance jumped to 0.1951.

5.3. Time-Weighted Frequency (Hit Rate @10: 0.3097)

Next, we used an exponential decay to weight recency:

$$\text{score}(p, c) = \sum_{t \in T_c} \mathbb{1}[\text{product}_t = p] \cdot e^{-\lambda(d_{\text{cutoff}} - d_t)},$$

where we took $\lambda = 0.1$ and d_t is the transaction date. This method aligned well with cyclical behaviors identified in EDA, producing a more sizable jump to a 0.3097 Hit Rate @10.

5.4. Hybrid (Freq. + Recency + Seasonal) (Hit Rate @10: 0.3559)

Incorporating seasonality, we model:

$$\text{score}(p, c) = 0.4 \cdot F(p, c) + 0.5 \cdot R(p, c) + 0.1 \cdot S(p, m),$$

where $F(p, c)$ is frequency-based, $R(p, c)$ captures recency, and $S(p, m)$ encodes monthly seasonal fluctuations (e.g., month $= m$). Observations from EDA, specifically the strong March spikes and Q4 surges, motivated this.

5.5. Hybrid (Freq. + Recency) (Hit Rate @10: 0.3584)

Here, we condense the signals to:

$$\text{score}(p, c) = 0.3 \cdot \frac{\text{freq}(p, c)}{\max_p \text{freq}(p, c)} + 0.7 \cdot \frac{\text{days}_{\max} - \text{days}_{\text{last}}(p, c)}{\text{days}_{\max}}.$$

We discovered that purely emphasizing frequency and recency, with weights at 30% and 70% respectively, yielded 0.3584. Which is very close to the seasonal hybrid approach, and it reflects how recency can be a potent predictor.

5.6. Customer Segmented (Hit Rate @10: 0.3605)

Recognizing the variance in purchasing frequencies, we segment customers and alter weights accordingly:

$$\text{score}(p, c) = w_f(s_c) \cdot F(p, c) + w_r(s_c) \cdot R(p, c),$$

where s_c is the segment label for customer c. The weighting scheme uses, for instance:

Weight Type	High	Medium	Low
Frequency (w_f)	0.25	0.35	0.45
Recency (w_r)	0.75	0.65	0.55

Table 4. Weighting Coefficients by Customer Segment

This deeper customization edged us to 0.3605, marking our best single-model performance.

5.7. Ensemble (Hit Rate @10: 0.3585)

Finally, an ensemble merges the top approaches:

$$\text{score}(p, c) = \sum_{i=1}^{4} w_i \cdot \text{score}_i(p, c),$$

with optimized weights:

Weight Type	Value
w_{category}	0.527
w_{temporal}	0.231
$w_{\text{segmented}}$	0.160
w_{hybrid}	0.082

In our trials, the ensemble hovered around 0.3585, just shy of the specialized segmented method.

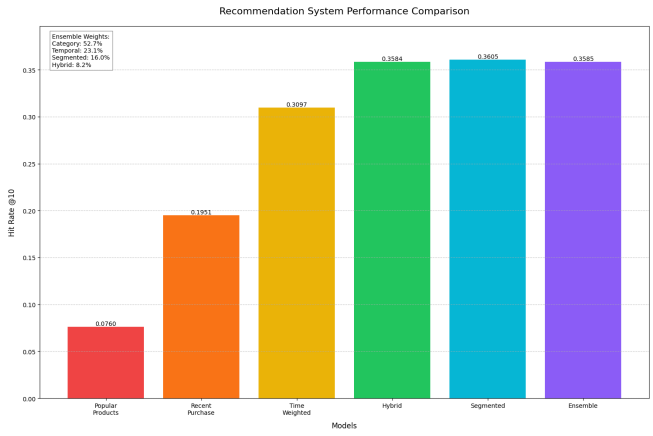

Figure 6. Model Performance by Approach

Overall, we moved from a naive 0.0760 baseline to a customer-segmented strategy exceeding 0.3600. Clearly, advanced signals (e.g., recency, segmentation, and time-based patterns) are instrumental to better predictions. The narrow gap between the best hybrid methods and the ensemble also indicates that well-engineered features may be more crucial than adding further complexity to the model.

6. Neural Graph Collabroative Filtering

Collaborative filtering (CF) is a core technique in recommendation systems that predicts a user's preferences based on their past interactions and the preferences of similar users (Koren et al., 2021). Traditional CF approaches are divided into memory-based methods, which calculate similarity directly (e.g., cosine similarity), and model-based methods, which use mathematical models to identify patterns in user and item data. While effective, these methods often struggle to capture complex, indirect relationships (high-order collaborative signals), especially in sparse datasets. To overcome this limitation, Graph Collaborative Filtering (GCF) introduces graph structures, representing users and items as nodes and their interactions as edges. Using Graph Neural Networks (GNNs), GCF captures deeper connections by iteratively propagating and aggregating information across the graph, enabling the model to uncover more nuanced patterns (He et al., 2017; Wang et al., 2020).

Neural Graph Collaborative Filtering (NGCF) is a graph-based recommendation framework that incorporates high-order collaborative signals into the embedding generation process. Unlike traditional approaches that focus only in direct user-item interactions, NGCF uses a user-item interaction graph to propagate and aggregate information through multiple layers (Wang et al., 2019). By doing so, it uncovers indirect connections between users and items, producing embeddings that are both richer and more meaningful. These features make NGCF especially effective for handling sparse and large-scale datasets, where capturing subtle collaborative signals is key to improving performance.

7. Methodology

This project adapts the Neural Graph Collaborative Filtering approach framework (Wang et al., 2019) for recommending supermarket products.The approach consists of four main components. First, a graph is created where nodes represent individual customers and products, with edges representing interactions weighted by the quantity purchased. Second, an embedding initialization step assigns random latent vectors to the nodes, this has the goal to be the starting point for learning user and product representations. Third, the model includes propagation layers that iteratively aggregates information from connected nodes, capturing deeper relationships within the graph. Finally, the model is trained using Bayesian Personalized Ranking (BPR) loss, ensuring the embeddings accurately reflect user preferences. The goal of this approach is to improve recommendation accuracy by capturing both direct and indirect relationships.

7.1. Embedding Layers

In this project, customer and product embeddings are initialized as random latent vectors, forming the starting point for the representation learning process. Each customer u and product i is associated with an embedding vector $e_u \in \mathbb{R}^d$ and $e_i \in \mathbb{R}^d$, where $d = 64$ is the embedding dimension. These embeddings are organized into a parameter matrix E, represented as:

$$E = \left[e_{u_1}, \ldots, e_{u_N} \quad | \quad e_{i_1}, \ldots, e_{i_M} \right],$$

where N and M are the total numbers of customers and products, respectively. This matrix has the purpose of being similar to a lookup table, with embeddings randomly initialized and optimized during training to capture meaningful latent features.

This initialization strategy is considered a standard practice in collaborative filtering, such as in Matrix Factorization (MF) (Koren et al., 2009) and Neural Collaborative Filtering (NCF) (He et al., 2017), where embeddings start as learnable parameters. By initializing embeddings randomly, the model can adaptively learn representations that reflect the underlying collaborative signal in the data. This process ensures that embeddings for customers and products provide a strong foundation for subsequent refinement through graph-based propagation layers.

7.2. Embedding Propagation Layers

The embedding propagation layers refine the embeddings of customers and products by including information from the interaction graph. These layers work iteratively, passing information between connected nodes, enabling the model to capture collaborative filtering (CF) signals at different levels of connectivity. With multiple propagation layers stacked together, the model is able to learn both direct (first-order) and more complex (high-order) relationships.

7.2.1. First-Order Propagation

The first-order propagation layer captures direct interactions between customers and products. For a connected pair (u, i), the message passed from product i to customer u is defined as:

$$m_{u\leftarrow i} = \frac{1}{\sqrt{|N_u||N_i|}} \left(W_1 e_i + W_2(e_i \odot e_u)\right),$$

where e_u and e_i are the embeddings of the customer and product, $\odot$ denotes element-wise multiplication, and W_1, W_2 are trainable weight matrices.

The messages are aggregated to refine the embeddings. For a customer u, the updated embedding is:

$$e_u^{(1)} = \text{LeakyReLU}\left(m_{u\leftarrow u} + \sum_{i\in N_u} m_{u\leftarrow i}\right),$$

where $m_{u\leftarrow u} = W_1 e_u$ is the self-message. This process combines the customer's original embedding with information propagated from connected products. The *LeakyReLU* activation function makes sure that small negative signals contribute to the embedding refinement, improving representation capacity.

7.2.2. High-Order Propagation in NGCF

To capture high-order connectivity in a user-item interaction graph, multiple embedding propagation layers are stacked. These layers allow users and items to aggregate information not just from their direct neighbors (first-order connections), but also from nodes several hops away, enriching the embeddings with collaborative signals.

At the l-th layer, the representation of a user u is updated recursively by combining messages from itself and its neighbors:

$$e_u^{(l)} = \text{LeakyReLU}\left(m_{u\leftarrow u}^{(l)} + \sum_{i\in\mathcal{N}_u} m_{u\leftarrow i}^{(l)}\right),$$

where:

- $m_{u\leftarrow u}^{(l)} = W_1^{(l)} e_u^{(l-1)}$ is the self-contribution message.
- $m_{u\leftarrow i}^{(l)} = p_{ui}\left(W_1^{(l)} e_i^{(l-1)} + W_2^{(l)}\left(e_i^{(l-1)} \odot e_u^{(l-1)}\right)\right)$ aggregates messages from neighboring items i.

Here, $W_1^{(l)}$ and $W_2^{(l)}$ are trainable transformation matrices, and $e_i^{(l-1)}$ is the embedding of item i from the previous layer. The operator $\odot$ performs element-wise multiplication, capturing interactions between embeddings. This process encodes collaborative signals, such as multi-hop interactions $(u_1 \leftarrow i_2 \leftarrow u_2 \leftarrow i_4)$, into the embeddings. To efficiently compute updates for all users and items at the same time, the propagation rule is expressed in matrix form:

$$E^{(l)} = \text{LeakyReLU}\left((L+I)E^{(l-1)}W_1^{(l)} + L\left(E^{(l-1)} \odot E^{(l-1)}\right)W_2^{(l)}\right),$$

where:

- $E^{(l)} \in \mathbb{R}^{(N+M)\times d_l}$ represents the embeddings of all users and items at layer l.
- L is the Laplacian matrix, calculated as:

  $$L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}},$$

  where A is the adjacency matrix and D is the degree matrix.
- I is the identity matrix.
- A is formed from the user-item interaction matrix R as:

  $$A = \begin{bmatrix} 0 & R \\ R^\top & 0 \end{bmatrix}.$$

Stacking multiple propagation layers allows the model to capture deeper and more complex collaborative signals,improving the quality of user and item embeddings. This method is especially useful for sparse datasets, where direct connections alone might not provide enough information.

7.3. Model Training and Prediction

This section explains the model's training process, beginning with how the adjacency matrix is built and the negative sampling strategy. It includes the training loop, the use of Bayesian Personalized Ranking (BPR) loss, optimization methods, and stopping criteria. Finally, it describes how the learned embeddings are used to make the final predictions.

7.3.1. Adjacency Matrix

The adjacency matrix A encodes the customer-product interaction graph, where each entry A_{ij} corresponds to the interaction strength between customer i and product j, defined as:

$$A_{ij} = \begin{cases} \text{weight}_{ij}, & \text{if } (i,j) \in E, \\ 0, & \text{otherwise}, \end{cases}$$

where E is the set of edges (interactions) in the graph, and weight_{ij} is the quantity purchased by customer i for product j. The adjacency matrix is symmetric, updated as $A = A + A^\top$, to capture bidirectional relationships in the graph.

Given that in the training dataset, a customer interacts with 30-50 products on average, the resulting matrix has a sparsity level greater than 99%. To reduce sparsity and introduce meaningful hardness, a category-based negative sampling strategy was created. For each positive interaction, a random negative instance from the same product category (e.g., *shelf level 1*) is selected. This helps the model train on harder examples, improving its ability to distinguish between similar but non-interacted items (He et al., 2017).

7.3.2. Optimization

The training process optimizes the Bayesian Personalized Ranking (BPR) loss, which has been extensively used in recommender systems for its ability to model pairwise preferences (Rendle et al., 2012). The BPR loss encourages the predicted preference for a purchased item i^+ to be higher than that for a negative item i^-. The loss is defined as:

$$\mathcal{L}_{BPR} = -\sum_{(u,i^+,i^-)} \ln \sigma\left(\hat{y}_{u,i^+} - \hat{y}_{u,i^-}\right) + \lambda \|\Theta\|^2,$$

where:

- $\sigma(\cdot)$ is the sigmoid function,
- $\hat{y}_{u,i}$ is the predicted preference score for customer u and item i,
- $\Theta = \{E, \{W_1^{(l)}, W_2^{(l)}\}_{l=1}^{L}\}$ includes all trainable model parameters, and
- λ is the L2 regularization term to prevent overfitting.

The model parameters are updated using the Adam optimizer with mini-batches of randomly sampled triples (u, i^+, i^-).

7.3.3. Dropout Techniques

To reduce overfitting, NGCF uses two types of dropout: message dropout and node dropout. Dropout has been widely used in deep learning to enhance model generalization (Srivastava et al., 2014). Message dropout randomly removes outgoing messages during propagation, reducing sensitivity to single connections between customers and products. This makes the learned embeddings less dependent on specific edges, making the model more robust to small changes in the graph structure. Node dropout randomly skips some nodes, preventing them from sending any messages during training. By doing so, it reduces the influence of individual users or products and encourages the model to identify patterns that generalize across the dataset.

Message dropout is applied with a probability p_1, affecting messages propagated in each layer. Node dropout, with probability p_2, randomly removes $(M + N)p_2$ nodes from the graph's Laplacian matrix in each layer, where M and N are the numbers of customers and products. These dropout techniques enhance the robustness of the learned embeddings by encouraging generalization.

7.3.4. Model Prediction

After the training process, the final embeddings for users (e_u^*) and items (e_i^*) are created by combining the outputs from all the model's propagation layers. Each layer produces embeddings that capture different levels of connectivity in the graph. The zero-th layer ($e_u^{(0)}$ or $e_i^{(0)}$) represents the initial embeddings, while subsequent layers ($e_u^{(1)}$, $e_u^{(2)}$, ..., $e_u^{(L)}$) refine these embeddings by incorporating increasingly higher-order interactions.

To combine these representations, NGCF uses a concatenation operation ($\|$), where embeddings from all layers are joined into a single vector:

$$e_u^* = e_u^{(0)} \| e_u^{(1)} \| \cdots \| e_u^{(L)}, \quad e_i^* = e_i^{(0)} \| e_i^{(1)} \| \cdots \| e_i^{(L)}.$$

This concatenation retains information from each layer, allowing the final embeddings to reflect both direct and indirect relationships in the graph. As result, the model can capture user preferences and item characteristics across multiple levels of connectivity.

Once the final embeddings are obtained, the preference score $\hat{y}_{NGCF}(u, i)$ between a user u and an item i is computed using an inner product:

$$\hat{y}_{NGCF}(u, i) = e_u^* \cdot e_i^*,$$

where e_u^* and e_i^* are the concatenated embeddings of the user and item, respectively. The inner product measures the similarity between the user and item in the learned embedding space, with higher scores indicating stronger preferences.

By utilizing a simple inner product for preference prediction, the model focuses on learning high-quality embeddings through the propagation layers.

7.4. LightGCN

Given the density of our user-item interaction data (87M interactions), we implemented LightGCN (He et al., 2020) to create an efficient and scalable recommendation system. Our implementation follows a simplified graph convolution approach that removes unnecessary feature transformation and nonlinear activation components while preserving the essential neighborhood aggregation.

7.4.1. Model Architecture

The core propagation rule in our implementation is defined as:

$$e_u^{(k+1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u| \cdot |\mathcal{N}_i|}} e_i^{(k)} \tag{1}$$

where $e_u^{(k+1)}$ represents the user embedding after the $(k+1)$-th layer, and $\mathcal{N}_u$ denotes the set of items interacted with by user u. The final embeddings are computed through layer combination:

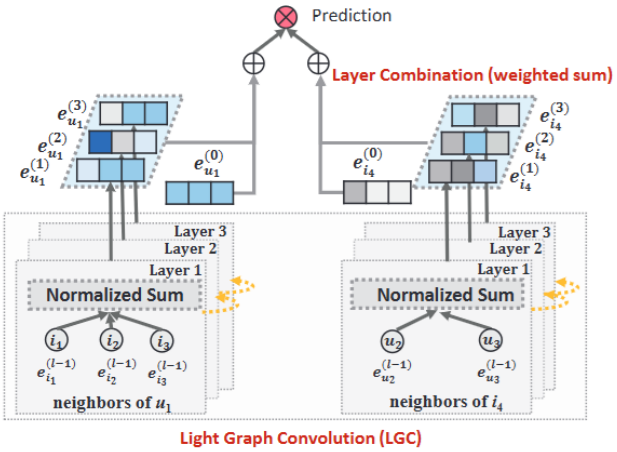

Figure 7. LightGCN Architecture

$$e_u = \sum_{k=0}^{K} \alpha_k e_u^{(k)} \tag{2}$$

with $\alpha_k = \frac{1}{K+1}$ as the layer weights, where $K = 3$ is our chosen number of propagation layers.

7.4.2. Training Strategy

We employed the Bayesian Personalized Ranking (BPR) loss for model optimization:

$$\mathcal{L}_{BPR} = \sum_{(u,i,j)} -\ln \sigma(\hat{y}_{u,i} - \hat{y}_{u,j}) + \lambda \|\Theta\|^2 \tag{3}$$

where (u, i, j) represents user u, positive item i, and negative item j. The implementation specifics include:

- Batch size of 5120 to maximize GPU utilization
- Adam optimizer with learning rate 0.001
- L2 regularization weight $\lambda = 1e-4$
- Random negative sampling with uniform distribution

7.4.3. Memory Optimization

Given our large-scale dataset (37.04 GB), we implemented several memory optimization techniques:

Table 5. Memory Optimization Techniques

Component	Optimization
Graph Storage	Sparse tensor format
Batch Processing	Chunk size: 4000 users
Embedding Updates	Layer-wise propagation
Evaluation	800 users per batch

7.4.4. Results and Performance Analysis

The LightGCN model demonstrated consistent performance improvements across multiple metrics during training. The training process, executed on system with 32GB RAM and 8GB GPU, revealed several key performance characteristics:

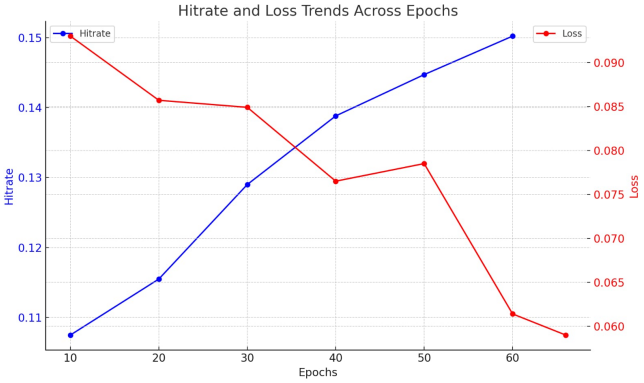

Figure 8. LightGCN Performance

The loss convergence pattern shows three distinct phases: rapid initial improvement (epochs 1-10, 0.179 to 0.107), moderate improvement (epochs 11-50, 0.106 to 0.065), and fine-tuning (epochs 51+, 0.064 to 0.049). This pattern suggests effective knowledge capture from the interaction graph.

7.5. Light Gradient Boosting Machine (LightGBM)

7.5.1. Introduction to LightGBM

LightGBM (Ke et al., 2017) is a high performance gradient boosting framework due to its efficiency, accuracy, and interpretability. For those previous reasons, we chose to use it on our large dataset. It is designed to handle large-scale datasets with faster training speeds by using techniques such as histogram-based learning.

7.5.2. Key Features

One of the most prominent feature of LightGBM is its leaf-wise decision tree growth shown in 9. The tree grows by adding a leaf that provides the maximum gain at each expansion step. Additionaly, it uses two techniques, Gradient-based One Side Sampling (GOSS) and Exclusive Feature Bundling (EFB), to overcome the limitations of the histogram-based approach used in Gradient Boosting Decision Tree (GBDT).

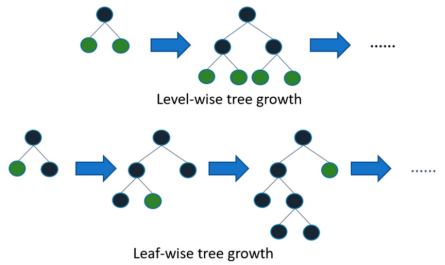

Figure 9. Level wise vs Leaf Wise tree growth

7.5.3. Mathmetical Foundations

LightGBM is an implementation of (GBDT) algorithm. It creates decision trees iteratively by minimizing a loss function. The objective at each iteration is to optimize the following loss function:

$$\mathcal{L}(\Theta) = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)} + \lambda f(x_i; \Theta_t)) + \Omega(f),$$

where $L(y_i, \hat{y}_i)$ is the error between predicted and actual labels, $\lambda f(x_i; \Theta_t)$ represents the contribution of the current tree, and $\Omega(f)$ regularizes the model to prevent overfitting.

7.5.4. LightGBM in recommendation systems

In recommendation problems, LightGBM predicts the likelihood of an interaction in the following steps: First, a dataset is prepared with user-item pairs, where each pair is labeled either 1 (positive interaction) or 0 (negative interaction). Features are engineered to capture user-item relationships. The goal is to predict the probability $P(y_{ui} = 1|x_{ui})$, where y_{ui} is the interaction between the user u and the item i, and x_{ui} represents the set of features for the pair of users and objects.

7.5.5. Features and Scalibility of LightGBM

For the LightGBM model, we focused on using user features to capture interaction patterns. The feature set included attributes such as transactionCount, uniqueTransactionCount, totalQuantity, avgQuantity, promoRatio, daysSinceFirst, daysSinceLast, and purchaseInterval. These features provided information about user activities, purchase behavior, and engagement with promotions.

One of the strong aspects of LightGBM in this context was its ability to scale. Unlike all other models we tried that required us to train on a sample of the dataset due to the computational bottlenecks. LightGBM successfully trained on the entire dataset in matter of minutes. This made it a great choice for our problem.

7.5.6. Results and Observations

We evaluated the LightGBM model using the hitrate@10 metric. The model achieved only a score of **0.17223**. Note that this was much lower than the baseline model, which attained a hitrate@10 of 0.33. This suggest that the feature engineering and selection process may not have captured the relationships and patterns between the users and products, limiting LightGBM's predictive capabilities.

While the results indicate a poor performance, is ability to process the entire dataset still highlights its potential as a robust tool for large-scale recommendation tasks.

8. Comparative Results and Implications

Table 6 summarizes the Hit Rate @10 scores achieved by different methodologies. The statistical approaches, particularly the customer-segmented method, demonstrated superior performance with a Hit Rate @10 of 0.3605, surpassing the baseline score of 0.33. This success stems from its effective combination of frequency and recency signals, weighted differently across customer segments. The method's ability to adapt to varying purchase patterns proved crucial in the grocery e-commerce context, where product repurchase intervals show high variability.

Table 6. Performance Comparison of Implementation Approaches

Method	Hit Rate @10
Statistical (Customer Segmented)	0.3605
Baseline	0.3300
LightGBM	0.1722
LightGCN (10% data)	0.1500
NGCF (1% data)	0.1240

The deep learning approaches, despite their sophisticated architectures, showed lower performance when compared to both the baseline and statistical methods. LightGCN, trained on 10% of the dataset, achieved 0.15, while NGCF, implemented with just 1% of customers, reached 0.124. These results suggest that in this grocery domain, the complexity of neural architectures may not inherently guarantee better recommendations, especially with limited data.

Similarly, LightGBM recorded a Hit Rate @10 of 0.1722, failing to surpass even the baseline level of 0.33. While LightGBM is generally efficient, the features that were engi-

neered did not sufficiently capture the intricate relationships between customers and products in the grocery setting.

Ultimately, these findings emphasize that domain-aware statistical methods can outperform more complex approaches in certain contexts. The strength of the customer-segmented model illustrates the importance of understanding purchasing frequencies and adapting to varying repurchase patterns.

9. Conclusion

The implementation and evaluation of multiple recommendation approaches for Carrefour's e-commerce platform revealed several critical insights about the grocery e-commerce domain. While we successfully optimized data processing and improved computational efficiency, the relatively low Hit Rate @10 scores across all methods (best performing at 0.3605) indicate the fundamental challenge of predicting grocery repurchases. This performance gap suggests that current recommendation approaches, including sophisticated neural architectures, may not fully capture the complexity of grocery shopping behavior.

The superior performance of statistical approaches over neural methods raises important questions about the role of model complexity in recommendation systems. Despite the theoretical advantages of NGCF and LightGCN in capturing high-order relationships, their lower performance on partial datasets (0.124 and 0.15 respectively) suggests that the additional complexity might not translate to better predictions in the grocery domain. This finding challenges the common assumption that more sophisticated models inherently lead to better recommendations.

A significant limitation of our work lies in the data sampling for neural approaches, where computational constraints restricted NGCF to 1% and LightGCN to 10% of the dataset. This constraint makes it impossible to directly compare their full potential against statistical methods. Even LightGBM, despite processing the complete dataset, achieved only 0.17223, indicating that efficient processing alone does not guarantee better predictions.

The discovered transaction patterns - highly variable purchase frequencies and small basket sizes averaging 1.48 items - suggest that future work should focus more on understanding and incorporating these domain-specific characteristics rather than increasing model complexity. The mean reorder interval of 12.76 days, combined with the wide range of transaction frequencies, indicates that temporal dynamics play a crucial role that current models may not adequately address.

These findings emphasize the need for a fundamental shift in how we approach grocery e-commerce recommendations. Rather than focusing solely on architectural innovations, future research should prioritize methods that can better capture the irregular, category-dependent nature of grocery shopping while maintaining computational efficiency at scale.

Table 7. Contribution Table

Name	Contribution (%)
Zahir AHMAD	30
Ajejandro CARVAJAL	30
Amgad KHALIL	30
AI	10
Total	**100**

References

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 173–182, 2017.

He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., and Wang, M. Lightgcn: Simplifying and powering graph convolution network for recommendation. *arXiv preprint arXiv:2002.02126*, 2020.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Machine Learning*, 2017.

Koren, Y., Bell, R. M., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer*, 42(8): 30–37, 2009.

Koren, Y., Rendle, S., and Bell, R. Advances in collaborative filtering. In *Recommender Systems Handbook*, pp. 91–142. Springer, 2021.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

Scherpinski, F. and Lessmann, S. Personalization in E-Grocery: Top-N versus Top-k rankings. *arXiv preprint arXiv:2105.14599*, 2021.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Wang, X., He, X., Wang, M., Feng, F., and Chua, T.-S. Neural graph collaborative filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 165–174, 2019.

Wang, X., Jin, H., Zhang, A., He, X., Xu, T., and Chua, T.-S. Disentangled graph collaborative filtering. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1001–1010, 2020.