From Traditional to Cutting-Edge: An Analysis of Advanced Recommender System Architectures for Boels Rental

Alexandru Gliga
Department of Advanced Computing Sciences
Faculty of Science and Engineering
Maastricht University
Maastricht, The Netherlands

***Abstract*—In this study, we aimed to develop and evaluate various recommender system architectures to enhance item recommendations for active rental contracts at Boels Rental, a leading European rental company. By focusing on newly created contracts, the research aimed to facilitate personalized suggestions. We established a baseline using a Contract-based KNN algorithm and expanded our exploration to include advanced models such as LightFM, LightGCN, GRU4Rec, and SASRec. Our findings revealed that the incorporation of account features and time encoding negatively impacted the performance of the Contract-based KNN model. LightFM demonstrated robust performance, particularly with larger datasets. LightGCN struggled with the dataset's sparsity and binary nature, indicating a need for more sophisticated preprocessing. GRU4Rec excelled in capturing sequential patterns, showing strong performance and efficiency. SASRec underperformed, requiring more extensive tuning. Evaluating the models on the full dataset confirmed GRU4Rec as the top performer, demonstrating substantial superiority in MAP@5, Recall@5, and Hit@5. The study's insights suggest that GRU4Rec and LightFM can be considered for deployment at Boels Rental, while LightGCN and SASRec need significant adjustments for better applicability.**

I. Introduction

In recent years, the complexity of the relationship between users and items has stimulated the development of advanced models in the field of Recommender Systems. These models include graph neural networks (e.g., GCN [1], GraphSAGE [2], GAT [3]) and sequence-aware neural networks (e.g., RNNs [4], Transformers, and Self-Attention Mechanisms [5]). Such models have significantly enhanced recommendation capabilities, as detailed in recent reviews [6].

Boels Rental[1] recognizes the importance of leveraging these advanced algorithms to enhance customer experience. This paper centers on data provided by Boels, with a primary goal of building a Collaborative Recommender System tailored for the Dutch business customer segment. The collaboration between Maastricht University and Boels Rental is facilitated by the KE@Work program[2].

At Boels, customers establish rental *contracts* through various channels, including the official website, email, phone, or at Boels depots. These *contracts* consist of various *items*, and this study focuses on generating item suggestions for newly formed *contracts*. This approach supports personalized marketing strategies such as targeted email campaigns and tailored website content.

This research is unique as it addresses the challenge of generating recommendations for active *contracts*, thereby circumventing the typical user cold-start problem in recommender systems [7]. Typically, these *contracts* are small, comprising 1 to 5 distinct items (see Figure 19).

To address this challenge, we initially established a baseline using an intuitive and explainable K-Nearest Neighbors algorithm tailored for contracts (Contract-based KNN). This model can be seen as a simplified variation of the Weighted Session-based KNN (WSKNN) algorithm [8], where the *contracts* are considered as input sessions. Subsequently, we expanded our exploration to include a diverse range of algorithm architectures, selecting representative models from several prominent categories: LightFM (Matrix Factorization) [9], LightGCN (Graph Neural Networks) [10], Gru4Rec (Recurrent Neural Networks) [11], and SASRec (Transformers) [12]. Each model was evaluated to determine its effectiveness in the given scenario, leading to the formulation of several research questions:

- **RQ1:** What features positively influence the performance of the Contract-based KNN algorithm, as measured by MAP@5, Recall@5, and Hit@5?
  - **RQ1.1:** How do account features influence the performance of the Contract-based KNN algorithm?
  - **RQ1.2:** How does time encoding influence the performance of the Contract-based KNN algorithm?
- **RQ2:** What is the performance of the considered models for a range of embedding size and epoch parameters,

This thesis was prepared in partial fulfilment of the requirements for the Degree of Bachelor of Science in Data Science and Artificial Intelligence, Maastricht University. Supervisor(s): Francesco Barile (DACS), Roan Schellingerhout (DACS), Ellen Houbiers (Boels)

[1]Boels Rental: one of the largest rental companies in Europe, providing both equipment and specialist rentals. https://group.boels.com/en/

[2]KE@Work: an honors track of the Data Science and Artificial Intelligence bachelor's program at Maastricht University. https://www.maastrichtuniversity.nl/research/department-advanced-computing-sciences/education/keworkmarble-20/kework-information-0

using sampled training and validation data, as measured by MAP@5, Recall@5, and Hit@5?

- **RQ3:** What is the performance of the best-found variations of the considered models when trained and evaluated on the full dataset?
  - **RQ3.1:** When the best-found variations of the considered models are trained and evaluated on the full dataset, what is the training and recommendation generation time?
  - **RQ3.2:** How do MAP@5, Recall@5, and Hit@5 metrics change when the best-found variations of the considered models are trained and evaluated on the full dataset?

The evidence shows that no feature improved the performance of the Contract-based KNN model. Specifically, adding account characteristics (e.g., revenue cluster, NACE codes) and temporal encodings (e.g., month, week) did not enhance the model's recommendation accuracy.

In contrast, the GRU4Rec model, which uses Gated Recurrent Units to capture sequential patterns, outperformed the Contract-based KNN baseline. GRU4Rec's ability to create a contract latent representation led to more accurate and contextually relevant recommendations across all key metrics (MAP@5, Recall@5, and Hit@5). This highlights the effectiveness of simple sequence-aware models in handling the inherent structure of the data.

II. Related Work

The field of recommender systems has experienced significant advancements over the years, particularly with the integration of state-of-the-art algorithms capable of handling complex data. This section reviews the algorithm architectures considered in this research and highlights previous studies that have applied these algorithms in similar contexts [13].

A. Session-based KNN

The K-Nearest Neighbors (KNN) algorithm is widely used in recommendation systems due to its simplicity and effectiveness in capturing user-item interactions through similarity measures. Ludewig and Jannach [14] evaluated various session-based KNN methods and demonstrated their effectiveness in providing real-time recommendations for e-commerce. Their neighborhood-based methods excelled in terms of Hit Rate (HR@20) and Mean Reciprocal Rank (MRR@20) on the TMALL and RETAILR datasets.

B. Matrix Factorization

Matrix factorization has been enhanced by the introduction of models like LightFM, developed by Kula [15]. LightFM is a hybrid model that combines collaborative and content-based filtering, leveraging both user and item metadata along with traditional matrix factorization techniques. Jiang et al. [16] further advanced this approach with xLightFM, which outperformed state-of-the-art lightweight factorization methods in terms of prediction quality and memory efficiency.

C. Graph-Based Approaches

Graph Convolutional Networks (GCNs), introduced by Kipf and Welling [17], extend traditional convolutional neural networks to graph-structured data. He et al. [10] proposed LightGCN, which simplifies traditional GCNs by removing feature transformation and nonlinear activation functions. Empirical evaluations have shown that LightGCN achieves state-of-the-art performance, significantly improving metrics such as Recall@20 on datasets like Amazon Book, with a 17.92% improvement compared to NGCF.

D. Sequence-Aware Models

Sequence-aware models have been developed to address the limitations of traditional Recurrent Neural Networks (RNNs). Hidasi et al. [11] introduced GRU4Rec, which employs Gated Recurrent Units (GRUs) for session-based recommendations. GRUs help mitigate the vanishing gradient problem by incorporating gating mechanisms that control the flow of information. GRU4Rec has shown substantial improvements in predicting the next item in a user's interaction sequence, outperforming methods such as POP, S-POP, Item-KNN, and BPR-MF. For instance, GRU4Rec achieved a HitRate@20 of 0.6322, marking a 24.82% improvement on the RSC15 dataset[3].

Kang and McAuley [12] developed the Self-Attentive Sequential Recommendation (SASRec) model, which leverages the attention mechanism to focus on the most relevant parts of a user's interaction history. Empirical results indicate that SASRec achieves state-of-the-art performance in sequential recommendation tasks. For example, on the Amazon Beauty dataset, SASRec improved NDCG@10 by 25.9% compared to neural models and 6.6% compared to non-neural methods.

III. Data

In this section, we provide an overview of the data that is collected from the Boels' internal databases. Understanding the structure of the data is essential for informing the choice of appropriate algorithms that are presented in the Section IV.

The data can be categorized into four main types: account data, item data, accessory data, and contract data. Each data type serves a specific purpose in the recommendation process and contributes unique information to build comprehensive contract and item profiles. The structure and attributes of each type are detailed below.

A. Account Data

Account data represents customer-related information, helping identify specific characteristics, demographic details, and segmentation of each user. The key attributes of account data include: *account_id*, *name*, *billing_city*, *billing_country*, *nace_code_lvl_1* to *nace_code_lvl_4*, *rfm_segment*, *revenue_cluster*, and *revenue_segment*.

[3]https://www.kaggle.com/datasets/chadgostopp/recsys-challenge-2015

B. Item Data

Item data provides essential details regarding the products offered by Boels. The attributes of items include: *item_id*, *name*, *description*, *product_code*, and *analysis_code*.

C. Accessory Data

Accessory data provides details about the accessories related to the primary products offered by Boels. These accessories are often essential for the operation or enhancement of the main products. The attributes of accessories include: *parent_id*, *accessory_id*, *country_code*, and *is_mandatory*.

D. Contract Data

Contract data establishes the relationship between accounts and items through the record of transactions, helping understand customer purchase behavior. The attributes include: *contract_id*, *account_id*, *item_id*, *time*, and *depot_country_code*.

For a detailed definition of all the columns presented before, refer to Appendix E. Additionally, consider examining the Appendix B, Appendix C, and Appendix D to visualize data distribution.

This structured data allows the building of a hybrid recommender system to understand customer preferences, recognize patterns, and deliver personalized recommendations.

IV. Methodology

This section outlines the methodology employed to develop and evaluate the models considered in this paper. We begin with a data quality analysis to verify the integrity of the input data, followed by data preprocessing steps to make the raw data suitable for modeling. Next, we describe the data split logic to partition the dataset into training, validation, and testing subsets. Subsequently, we detail the implementation of five different recommender system algorithms: Contract-based KNN, LightFM, LightGCN, GRU4Rec, and SASRec.

In this research, we focus on generating the top 5 recommendations for each contract. This number was determined through discussions with the data science team, aiming to provide customers with a manageable number of recommendations while still offering a reasonable range of choices.

A. Data Quality Analysis

1) Account Data: The total number of records in the account data table is 201071. Initial analysis reveals missing values in several columns, and the percentages of accounts having NaN entries are displayed in the Table I.

2) Item Data: The initial number of items in the dataset is 32917. After removing duplicates based on the *item_id* column while keeping the item with the fewest missing values for the name, description, product_code, and analysis_code columns, we get 3810 records in total. The analysis reveals missing values in 4 different columns, and the percentages of NaN entries are displayed in the Table II.

3) Accessory Data: 73835 of *parent_id* - *item_id* pairs, with no missing values. After removing the duplicate rows, we end up with 73453 of *parent_id* - *item_id* pairs.

TABLE I
Missing Data Analysis for Account Attributes

Attribute	Percentage of Missing Values (%)
account_id	0.0676
name	0.0000
billing_city	0.0024
billing_country	0.0000
nace_code_lvl_1	21.6724
nace_code_lvl_2	21.6724
nace_code_lvl_3	21.9086
nace_code_lvl_4	22.0504
rfm_segment	64.2479
revenue_segment	63.7511
revenue_cluster	63.7511

TABLE II
Missing Data Analysis for Item Attributes

Attribute	Percentage of Missing Values (%)
item_id	0.0000
name	19.3438
description	29.4488
product_code	17.6377
analysis_code	26.0367

4) Contract Data: In the case of contract data, we extract the Dutch records from 2021 to 2023 (both inclusive). We do not include the earlier years because they are archived and limited records are available in the database. We can visualize the number of rented items per year in Figure 13. The contract data consists of 14267904 entries, this number signifying the count of rented items by business customers between 2021 and 2023 (both inclusive). There is no missing contract data.

B. Data Preprocessing

Following this, we handle the missing data and transform the original problem into a classification task. For that, we remove the duplicate *contract_id* and *item_id* pairs, ending up with 3996621 rented unique items per contract. We further reduce the size of the contracts by removing mandatory and optional accessories. The resulting distribution of contract sizes can be visualized in Figure 1. We continue by removing the contracts that have less than 2 items, ending up with 276879 contracts, which include 822683 unique items in all contracts.

Next, we filter the account data, such that it includes only business customers that created at least one of the above-filtered contacts. We receive 34733 accounts that have missing data, shown in the Table III.

We follow the same procedure and filter items, such that we end up only with items that are presented in at least one contract. The number of remaining items is 2843, with missing data presented in the Table IV.

Following, we remove the contracts containing NaN values and items that do not have a description. This shrinks the contracts to 478854 of rented items. Again, we remove the contracts with less than 2 rented items, reducing the data to 393375 contract item pairs. This is the last preprocessing step that involves the contracts. Additionally, we use the unique

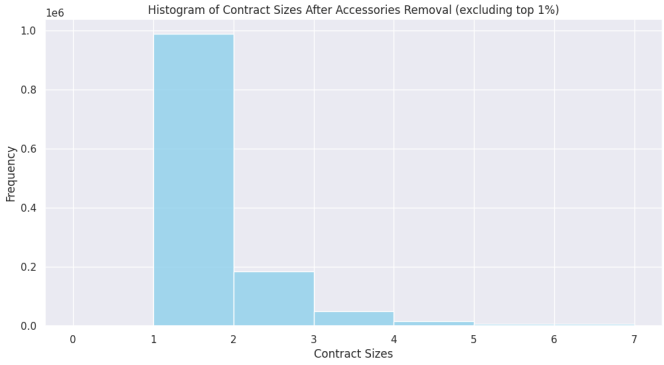

Fig. 1. Histogram of Contract Sizes After Accessories Removal (excluding top 1%).

TABLE III
MISSING DATA ANALYSIS FOR ACCOUNT ATTRIBUTES AFTER CONTRACTS FILTERING

Attribute	Percentage of Missing Values (%)
account_id	0.0000
name	0.0000
billing_city	0.0024
billing_country	0.0000
nace_code_lvl_1	3.8752
nace_code_lvl_2	3.8752
nace_code_lvl_3	3.9818
nace_code_lvl_4	4.3676
rfm_segment	1.2034
revenue_segment	1.1717
revenue_segment	1.1717

items and accounts to filter their associated datasets, ending up with 2041 items and 26355 accounts.

C. Data Split Logic

The number of cleaned contracts is consistent among year-month pairs (Figure 2).

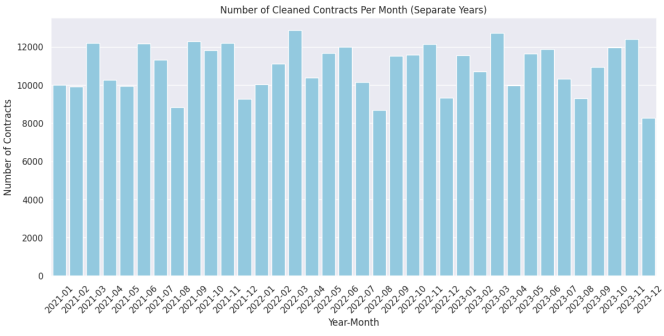

Fig. 2. Number of Cleaned Contracts Per Month (Separate Years).

We utilize the first two years of data for model parameter tuning and validation. The data from 2023 is reserved for testing. Due to the extensive volume of data and the limitations of our processing units, we create a representative sample for training and validation. Using the monthly expanding window validation method (for a visual explanation, see Figure 20), we ensure that our model is properly validated.

TABLE IV
MISSING DATA ANALYSIS FOR ITEM ATTRIBUTES AFTER CONTRACTS FILTERING

Attribute	Percentage of Missing Values (%)
item_id	0.0000
name	17.3760
description	26.5564
product_code	0.0000
analysis_code	10.6225

We obtain a significant subset of the validation data with respect to the individual months presented in the dataset. For that, we consider the month with the least number of contracts (8284), use the calculator[4] with a 99% significance level and 5% margin of error and obtain 614. So, by sampling the 0.074 fraction of all the months, we maintain the previously mentioned significance level and margin of error.

D. Algorithms Implementation

In this subsection, we detail the implementation of five different recommender system algorithms: Contract-based KNN, LightFM [9], LightGCN [10], GRU4Rec [11], and SASRec [12]. Each of these models has the potential to leverage the structured data described in the previous sections to generate semi-personalized recommendations for Boels' business customers.

Due to time constraints, we focused on optimizing the embedding size and training epoch parameters. All other hyper-parameters were set to their default values, as specified in widely used implementation libraries and reference works: LightFM [9], LightGCN [18], GRU4Rec [19], and SASRec [19]. For a detailed definition of the hyper-parameters, refer to Appendix F.

Contract-Based K-Nearest Neighbors (KNN): The model is a memory-based approach that utilizes the similarity between contracts to generate recommendations:

- *Feature Construction:* We construct a contract-item matrix from the transactions data. Every row represents a unique contract and every column represents a unique item. Each entry represents a binary number, 1 if the item is rented in the contract, 0 otherwise. For example, let us say that we have 2 contracts and 3 items in the system, and the following composition: *contract_1 = [item_1, item_2]* and *contract_2 = [item_2, item_3]*. We can represent this using the Table V.

TABLE V
EXAMPLE OF CONTRACT ITEMS TABLE

	item_1	item_2	item_3
contract_1	1	1	0
contract_2	0	1	1

We also make possible augmentation of account features and their corresponding contracts. We can express the

[4]https://www.qualtrics.com/blog/calculating-sample-size/

contract row using the one-hot encoding of one or more categorical features of the account that created the contract. For example, let us use the data from the previous example, and, additionally, consider that the account that created *contract_1* is part of the *C (construction)* industry and the account that created *contract_2* is part of the *A (agriculture)* industry. We can represent this using the Table VI.

TABLE VI
EXAMPLE OF CONTRACT ITEMS TABLE WITH AUGMENTED ACCOUNT DATA

	item_1	item_2	item_3	industry_C	industry_A
contract_1	1	1	0	1	0
contract_2	0	1	1	0	1

- *Similarity Metric:* We employ Jaccard similarity [20] metrics to quantify the closeness between vectors in our contract-item matrix. This choice is made because of its suitability and explainability for the binary encoding. The Jaccard similarity coefficient is defined as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

where A and B are the sets of items in two contracts, $A \cap B$ is the intersection of the sets (items present in both contracts), and $A \cup B$ is the union of the sets (items present in either or both contracts).
- *Recommendation Generation:* For a given contract, the algorithm identifies the k-nearest neighbors using Jaccard similarity. Each neighbor is unique, ensuring no two contracts in the neighborhood share the same item set. It computes item scores by multiplying each item's frequency (always 1 if the item is present in the contract, 0 otherwise) in the neighbors by the respective similarity scores. These scores are summed up across all neighbors. The top items with the highest aggregated scores are then recommended.

This approach is particularly useful for capturing explicit relationships and providing interpretable recommendations based on transactional history and account features. The only tunable parameter in the model's implementation is the one that determines the size of the nearest neighbourhood - *num_neighbours*. Given our implementation, we set it to 5, which will generate at least 5 distinct item recommendations in the majority of the cases.

LightFM: LightFM is a hybrid matrix factorization model that incorporates both user and item metadata into the recommendation process. This model is well-suited to handle sparse datasets.

- *Model Structure:* LightFM combines the advantages of collaborative filtering and content-based methods by using latent embeddings for users (in our case contracts) and items, enriched with metadata embeddings from the account and item data. We represent our contracts using the one-hot encoding of the account features, which created the contract. In the case of items, we represent them by embedding their description using the all-MiniLM-L6-v2 model[5] (state-of-the-art sentence transformer).
- *Learning Process:* We train the LightFM model using Weighted Approximate-Rank Pairwise (WARP) loss function, which is well-suited for binary interaction data [9].
- *Recommendation Generation:* The LightFM model generates recommendations by computing the dot product between the learned contract and item latent vectors.

Also, we will consider 2 variations of the LightFM model:

- **LightFM Standard**: model built using transactions data exclusively.
- **LightFM Hybrid**: model built using transactions, account, and item embeddings.

For a detailed definition of the model's hyper-parameters, refer to Appendix F1.

LightGCN: The Light Graph Convolutional Network (LightGCN) simplifies the architecture of traditional graph convolutional networks by removing feature transformation and nonlinear activation layers. This design enhances computational efficiency and improves performance in recommendation tasks.

LightGCN, along with another variation known as EmbGCN, achieved state-of-the-art results on the Amazon-Book dataset in 2020 and 2021[6]. This model is still actively researched and remains a prominent choice for tasks in recommender systems and collaborative filtering[7].

- *Graph Construction:* We construct a bipartite graph from our contract-item binary data, where nodes represent contracts and items, and edges represent binary transactions (rentals).
- *Embeddings Generation:* LightGCN generates embeddings for both contracts and items by iteratively aggregating information from neighboring nodes. Initially, each contract and item is assigned a unique embedding vector. Through multiple layers of graph convolution, each node's embedding is updated by taking the mean of its neighbors' embeddings. This process captures the structural information and interactions within the graph, leading to meaningful contract and item representations.
- *Recommendation Generation:* Recommendations are generated by calculating the inner product between the learned embeddings of contracts and items. For a given contract, the inner product scores with all item embeddings are computed, and items are ranked based on these scores. The items with the highest scores are recommended, as they are considered the most relevant to the contract based on the learned embeddings.

For a detailed definition of the model's hyper-parameters, refer to Appendix F2.

[5]https://deepinfra.com/sentence-transformers/all-MiniLM-L6-v2
[6]https://paperswithcode.com/sota/recommendation-systems-on-amazon-book
[7]https://paperswithcode.com/method/lightgcn

GRU4Rec: The Gated Recurrent Unit for Recommendations (GRU4Rec) leverages the power of recurrent neural networks (RNNs), specifically GRUs, to model sequential data in recommendation systems. GRU4Rec was one of the pioneering methods to apply RNNs to recommender systems, achieving significant improvements in performance for session-based recommendations. The model's architecture and training methodology have been continuously refined, making it a robust choice for sequence-aware recommendation tasks [11].

- *Event Representation:* GRU4Rec processes rented items in a contract as a sequence of events. Each event is encoded into a fixed-size vector, capturing relevant information about the event.
- *Sequence Modeling:* The encoded vectors are fed into a GRU-based RNN. The GRU cells are designed to capture dependencies between events in the sequence by maintaining a hidden state that is updated at each time step. This hidden state serves as a dynamic representation of the contract's purpose/meaning based on the sequence of items.
- *Recommendation Generation:* At each time step, GRU4Rec generates a probability distribution over all possible items using the hidden state of the GRU. The items are ranked based on these probabilities, with higher probabilities indicating a higher likelihood of the item being interacted with next. The top-ranked items are recommended, providing personalized and contextually relevant suggestions.

For a detailed definition of the model's hyper-parameters, refer to Appendix F3.

SASRec: The Self-Attentive Sequential Recommendation (SASRec) model utilizes self-attention mechanisms to capture the sequential patterns in contracts. This approach addresses the limitations of traditional recurrent models by leveraging the self-attention mechanism to directly model the dependencies between different positions in a sequence, providing a more flexible and powerful way to understand customer behavior.

SASRec has demonstrated state-of-the-art performance in various sequential recommendation tasks, showcasing its ability to effectively capture long-range dependencies and complex interaction patterns [12].

- *Sequence Representation:* SASRec treats a contract as an ordered list. Each item in the sequence is represented as an embedding vector, which encodes its characteristics.
- *Self-Attention Mechanism:* The core of SASRec is the self-attention mechanism, which computes attention scores for each pair of positions in the sequence. These scores determine the importance of each item relative to others in the sequence, allowing the model to focus on relevant relationships. The attention mechanism captures both short-term and long-term dependencies, making it effective for modeling customer behavior.
- *Recommendation Generation:* SASRec generates recommendations by predicting the next item in the sequence. The model uses the self-attention outputs to compute a probability distribution over all possible items, indicating the likelihood of each item being the next interaction. The items with the highest probabilities are recommended, providing personalized suggestions based on the entire sequence of items in the contract.

For a detailed definition of the model's hyper-parameters, refer to Appendix F4.

V. Experiments and Results

In this section, we present the experimental setup and results of a series of experiments for the various recommender system models implemented: Contract-based KNN, LightFM, LightGCN, GRU4Rec, and SASRec. We aim to address the research questions, focusing on model performance, feature influence, and training efficiency.

A. Experimental Setup

Each model is tested under the same conditions to ensure fair comparisons. We utilize the first two years of data (2021 and 2022) for model training and validation, employing an *expanding window strategy for monthly validation* (Figure 20) to closely mimic real-world applications and anticipate future performance. Also, the best-found variation of each model is trained on full 2021-2022 data and tested on data from 2023 using the hold-out method.

The performance of each model was assessed using the following metrics. The decision to use the top 5 recommendations was made by the data science team at Boels, aiming to provide customers with a manageable number of recommendations while still offering a reasonable range of choices:

- *MAP@5:* Measures the precision of the top 5 recommendations, averaged across all contracts.
- *Recall@5:* Measures the proportion of relevant items successfully recommended in the top 5.
- *Hit@5:* Measures the presence of at least one relevant item in the top 5 recommendations.

The training and testing of each model were conducted using the computational resources available on the Kaggle platform[8]. Specifically, the CPU and GPU configurations were as follows:

- *CPU:* Intel Xeon processors.
- *GPU:* NVIDIA Tesla P100 GPUs.

B. Feature Influence on Contract-based KNN

To answer **RQ1** we firstly need to quantify the influence of the account features (**RQ1.1**) and time encoding (**RQ1.2**) on the performance of the Contract-based KNN model. The Table VII summarizes the average metrics for different feature types.

[8]https://www.kaggle.com

TABLE VII
INFLUENCE OF FEATURES ON MAP@5, RECALL@5, AND HIT@5 METRICS FOR CONTRACT-BASED KNN MODEL

Feature Type	Feature Name	MAP@5	Recall@5	Hit@5
	No Features	**0.2325**	**0.4210**	**0.4388**
Account	Revenue Cluster	0.0201	0.0422	0.0450
Account	Revenue Segment	0.0262	0.0580	0.0639
Account	RFM Segment	0.0107	0.0177	0.0202
Account	NACE Code Level 1	0.0139	0.0261	0.0279
Account	NACE Code Level 2	0.0172	0.0484	0.0514
Account	NACE Code Level 3	0.0133	0.0341	0.0369
Account	NACE Code Level 4	0.0138	0.0293	0.0307
Time	Month	0.0095	0.0208	0.0214
Time	Week	0.0223	0.0449	0.0488
Time	Season	0.0149	0.0326	0.0343
Combination	Combination 1[1]	0.0048	0.0138	0.0161
Combination	Combination 2[2]	0.0143	0.0317	0.0350

[1] Revenue Cluster, NACE Code Level 1, Season
[2] Revenue Cluster, Revenue Segment, NACE Code Level 1, NACE Code Level 2, Season, Week

C. Model Performance Comparison

To answer **RQ2**, we evaluated each model's performance using the cleaned, pre-processed, and sampled dataset (2021-2022), focusing on MAP@5, Recall@5, and Hit@5 metrics. The parameters of the best-found variation of each considered model are as follows:

- *LightFM Standard*: 10 epochs and 50 embedding size;
- *LightFM Hybrid*: 50 epochs and 50 embedding size;
- *LightGCN*: 800 epochs and 32 hidden size;
- *GRU4Rec*: 5 epochs and 64 hidden size;
- *SASRec*: 5 epochs and 16 hidden size.

The performance of these models is included in Table VIII.

TABLE VIII
PERFORMANCE ON SAMPLED DATASET (METRICS)

Model	MAP@5	Recall@5	Hit@5
CB KNN	0.2325	0.4210	0.4388
LightFM Standard	0.1901	0.3757	0.3960
LightFM Hybrid	0.0580	0.1234	0.1310
LightGCN	0.0182	0.0404	0.0416
GRU4Rec	**0.3841**	**0.5374**	**0.5603**
SASRec	0.0307	0.0697	0.0772

A more detailed overview of the models' results is presented in Appendix H.

D. Training Time and Efficiency on Full Dataset

To fully answer **RQ3**, we firstly address **RQ3.1**. For that, we train the best-found variations of the considered models on the full dataset (2021-2022) and evaluate their performance in terms of training time, using the full test dataset (2023). The results are presented in Table IX.

To address **RQ3.2**, we use the process described in the previous step and evaluate their performance in terms of MAP@5, Recall@5, and Hit@5 metrics. The results are presented in Table X.

TABLE IX
PERFORMANCE ON FULL DATASET (TIME)

Model	Train Time	Rec Time
CB KNN[1]	169s	5 hours
LightFM Standard	79s	154s
LightFM Hybrid	139s	140s
LightGCN	3194s	297s
GRU4Rec	**28s**	**42s**
SASRec	49s	85s

[1] The CB KNN model ran for 5 hours and returned 511/53546 recommendations.

TABLE X
PERFORMANCE ON FULL DATASET (METRICS)

Model	MAP@5	Recall@5	Hit@5
CB KNN[1]	0.0306	0.0990	0.1039
LightFM Standard	0.2104	0.4219	0.4441
LightFM Hybrid	0.1779	0.4119	0.4330
LightGCN	0.0017	0.0040	0.0045
GRU4Rec	**0.3640**	**0.5039**	**0.5273**
SASRec	0.0138	0.0268	0.0311

[1] The CB KNN model ran for 5 hours and returned 511/53546 recommendations.

VI. DISCUSSION

The primary goal of this study was to develop and evaluate various recommender system models, built using different architectures, to enhance item recommendations for active rental contracts at Boels Rental. By focusing on newly created contracts, the study aimed to facilitate personalized marketing strategies. This section discusses the findings, implications, and potential improvements for the evaluated recommender systems.

A. Feature Influence on Contract-based KNN

Our investigation into the influence of features on the Contract-based KNN model (**RQ1**) provided several critical insights. The base model without any additional features showed reasonable performance, with MAP@5 at 0.2325, Recall@5 at 0.4210, and Hit@5 at 0.4388. However, incorporating account features such as revenue cluster and revenue segment decreased the model's performance. This suggests that these features may not align well with the binary nature of the contract-item matrix.

Time-based features showed mixed results. Weekly time encoding performed better on MAP@5, Recall@5, and Hit@5 metrics compared to monthly and seasonal encodings, but none outperformed the model without any additional features. Moreover, combining features did not yield performance gains. This indicates that treating all features equally in the similarity calculation might limit the model's ability to leverage the added information effectively. Future work should consider employing models where each feature has a learnable weight during similarity calculation to assign appropriate importance to each feature.

B. Influence of Model Parameters, Validated on Sampled Data

To address **RQ2**, we investigated the effects of embedding size and epoch parameters on the performance of various recommender models using sampled training and validation data. The choice of these specific hyperparameters is due to their

significant influence on model performance and training efficiency. Embedding size determines the dimensionality of the feature representation, directly impacting the model's ability to capture complex relationships. Epochs control the number of complete passes through the training dataset, affecting the model's convergence and generalization capabilities.

While other hyperparameters, such as learning rate, batch size, and regularization factors, can also have substantial impacts, we focused on embedding size and epochs due to time constraints and their primary role in model training. Future work could expand this analysis to include a broader range of hyperparameters to further optimize model performance.

LightFM: The results indicate that higher numbers of components generally improved the model's performance. Specifically, a combination of 10 epochs and an embedding size of 50 yielded the highest MAP@5, Recall@5, and Hit@5 metrics when features were not included. In the case of the hybrid model, the constantly increasing performance with the number of epochs and embedding size suggests that the model was underfitting the data.

LightGCN: LightGCN did not perform as expected on our dataset. The highest MAP@5 was achieved with an embedding size of 800 and 1000 epochs, but the overall performance remained low. This indicates that LightGCN may not be well-suited for the sparse, binary nature of our contract-item interactions. Alternatively, it may require more sophisticated preprocessing or feature engineering to enhance its effectiveness.

GRU4Rec: The model achieved the best results with a hidden size of 5 to 25 epochs, highlighting its ability to capture sequential patterns effectively. The model's performance improved consistently with increasing epochs up to a point. The highest dimension (64) showed the best result regarding the embedding size. This suggests that GRU4Rec still has room for improvement, and can effectively model the sequential nature of the rental contracts, making it a suitable choice for our recommendation task.

SASRec: Given its theoretical strengths in capturing long-range dependencies through self-attention mechanisms, SASRec underperformed compared to other models when applied to small item contract sequences. The best performance was observed during the first of 5 and 10 epochs, but the overall metrics were still lower than those achieved by GRU4Rec and LightFM. Interestingly, increasing the embedding dimensions worsened the performance. All of this suggests that while SASRec has potential, it may require more extensive tuning and optimization to generalize better to the data.

C. Performance and Efficiency of Different Models on Full Dataset

Evaluating the models on the full dataset to address **RQ3.1** and **RQ3.2**, we found:

Contract-Based KNN: Because a Kaggle session can run for a limited amount of time, the experiment was stopped after 5 hours, during which the Contract-Based KNN model generated recommendations for 511 contracts. A big factor in this weak time performance is the iterative approach in searching for the nearest neighbours. Another surprising result is that all metrics dropped compared to the previous experiment (MAP@5 to 0.0306, Recall@5 to 0.0990, Hit@5 to 0.1039). This suggests that the model overfitted the training data, and a possible solution is to train using a sampled dataset or increase the number of considered nearest neighbours.

LightFM: This model demonstrated robust performance, surpassing the results achieved in the experiments with sampled data. The standard implementation improved its metrics by over 10%, while the hybrid model saw an impressive increase of over 200%. Both models maintained relatively low training times (79 seconds for the standard model and 139 seconds for the hybrid model) and recommendation generation times (154 seconds for the standard model and 140 seconds for the hybrid model). This indicates that increasing the amount of training data enhanced overall performance without compromising feasibility.

LightGCN: This model struggled to deliver strong performance, both in the experiments with sampled data and the full dataset. Despite extensive training (3194 seconds), the model achieved a MAP@5 of only 0.0017, Recall@5 of 0.0404, and Hit@5 of 0.0416, indicating no improvement over the sampled data results. The recommendation generation time was relatively quick at 297 seconds, but the overall metrics remained low. This suggests that the LightGCN model may not be well-suited for the sparse, binary nature of the contract-item interactions in this dataset. Further sophisticated preprocessing or feature engineering might be required to enhance its effectiveness.

GRU4Rec: This model excelled in performance, relative to the other considered algorithms. The model's metrics showed substantial superiority, with a MAP@5 of 0.3640, Recall@5 of 0.5039, and Hit@5 of 0.5273. GRU4Rec maintained efficient training and recommendation times, training in just 28 seconds and generating recommendations in 42 seconds. This indicates that the GRU4Rec model effectively handled the sequential nature of the rental contracts, demonstrating strong capability and efficiency, making it a highly suitable choice for our recommendation task. Moreover, this is the only model that outperformed the baseline in both experiments.

SASRec: Despite its theoretical strengths, given the integrated self-attention mechanisms, SASRec underperformed compared to other models in this study. The model achieved a MAP@5 of 0.0138, Recall@5 of 0.0268, and Hit@5 of 0.0311, showing no improvement over the sampled data results. SASRec had efficient training and recommendation times, with training taking 49 seconds and recommendation generation taking 85 seconds. These results suggest it may require more extensive tuning and optimization to better handle the small item contract sequences present in this dataset.

D. Implications for Boels Rental

The findings from this study provide actionable insights for Boels Rental. Implementing a GRU4Rec model could enhance the accuracy and relevance of item recommendations for active

contracts. This has the potential not only to improve customer satisfaction but also to enable more effective personalized marketing strategies. Additionally, LightFM, with its ability to incorporate metadata, could serve as a robust alternative, particularly when leveraging additional item and account features (after additional parameter tuning), which opens room for explainability.

E. Limitations and Future Work

This study faced several limitations that impacted the performance of certain models. One of the primary limitations was the sparsity and binary nature of the dataset, which posed challenges for models like LightGCN and SASRec. These models struggled to effectively capture the interactions between contracts and items, resulting in suboptimal performance.

Additionally, the computational constraints of the Kaggle environment limited the ability to fully explore the parameter space for each model. Future research should aim to utilize more powerful computational resources to enable a more exhaustive search for optimal hyperparameters, especially for models that require extensive tuning.

Future work could also explore hybrid models that combine the strengths of different algorithms. For example, integrating GRU4Rec's ability to capture sequential patterns with LightFM's capability to incorporate metadata could potentially yield superior performance. Moreover, advanced techniques like meta-learning and transfer learning could be investigated to further improve model generalization.

Furthermore, conducting user studies with real customers to evaluate the recommendations in a beta version could provide valuable feedback. Enhancing model explainability is also crucial, as clear explanations for recommendations can increase user trust and engagement.

Finally, exploring other session-based RNN, Transformer, and GNN models could provide valuable insights into capturing complex interactions over time, which is particularly relevant for recommendation tasks involving sequences of user actions.

VII. Conclusions

The primary goal of this study was to develop and evaluate various recommender system models to enhance item recommendations for active rental contracts at Boels Rental.

Our investigation into the influence of features on the Contract-based KNN model revealed that incorporating account features and time encoding did not improve performance. The base model without additional features performed reasonably well on the sampled data, suggesting that the current feature set may not align well with the binary nature of the contract-item matrix. Future work should consider models that assign appropriate importance to each feature.

We also explored the effects of embedding size and epoch parameters on model performance using sampled data. LightFM demonstrated robust performance with improvements in metrics when more data was used. LightGCN struggled with the dataset's sparsity and binary nature, indicating a need for sophisticated preprocessing. GRU4Rec showed the best results, effectively capturing sequential patterns and demonstrating strong performance and efficiency. SASRec underperformed compared to other models, suggesting it requires more extensive tuning to handle small item contract sequences effectively.

Evaluating the models on the full dataset confirmed most of these findings. GRU4Rec remained the top-performing model with substantial superiority in MAP@5, Recall@5, and Hit@5. LightFM also showed robust performance with increased data, while LightGCN and SASRec continued to struggle with the dataset's characteristics.

In terms of efficiency and scalability, GRU4Rec and LightFM maintained efficient training and recommendation times, making them suitable for practical deployment. The Contract-based KNN model had impractical recommendation generation times.

Overall, addressing these limitations and pursuing these future directions could significantly advance the development of recommender systems for Boels Rental, leading to more effective and personalized recommendations.

References

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[2] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

[3] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[4] C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing, "Recurrent recommender networks," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, ser. WSDM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 495–503. [Online]. Available: https://doi.org/10.1145/3018661.3018689

[5] S. Zhang, Y. Tay, L. Yao, and A. Sun, "Next item recommendation with self-attention," *arXiv preprint arXiv:1808.06414*, 2018.

[6] Y. Li, K. Liu, R. Satapathy, S. Wang, and E. Cambria, "Recent developments in recommender systems: A survey," 2023.

[7] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong, "Addressing cold-start problem in recommendation systems," in *Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication*, ser. ICUIMC '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 208–211. [Online]. Available: https://doi.org/10.1145/1352793.1352837

[8] S. Moliński, "Wsknn-weighted session-based k-nn recommender system," *Journal of Open Source Software*, vol. 8, no. 90, p. 5639, 2023.

[9] M. Kula, "Metadata embeddings for user and item cold-start recommendations," in *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015.*, ser. CEUR Workshop Proceedings, T. Bogers and M. Koolen, Eds., vol. 1448. CEUR-WS.org, 2015, pp. 14–21. [Online]. Available: http://ceur-ws.org/Vol-1448/paper4.pdf

[10] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, 2020, pp. 639–648.

[11] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.

[12] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *2018 IEEE international conference on data mining (ICDM)*. IEEE, 2018, pp. 197–206.

[13] V. Maphosa and M. Maphosa, "Fifteen years of recommender systems research in higher education: Current trends and future direction," *Applied Artificial Intelligence*, vol. 37, no. 1, p. 2175106, 2023.
[14] M. Ludewig and D. Jannach, "Evaluation of session-based recommendation algorithms," *User Modeling and User-Adapted Interaction*, vol. 28, pp. 331–390, 2018.
[15] M. Kula, "Metadata embeddings for user and item cold-start recommendations," *arXiv preprint arXiv:1507.08439*, 2015.
[16] G. Jiang, H. Wang, J. Chen, H. Wang, D. Lian, and E. Chen, "xlightfm: Extremely memory-efficient factorization machine," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021, pp. 337–346.
[17] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[18] M. Labonne, *Hands-on Graph Neural Networks Using Python: Practical Techniques and Architectures for Building Powerful Graph and Deep Learning Apps with PyTorch*. Birmingham, UK: Packt Publishing Ltd., 2023.
[19] W. X. Zhao, Y. Hou, X. Pan, C. Yang, Z. Zhang, Z. Lin, J. Zhang, S. Bian, J. Tang, W. Sun, Y. Chen, L. Xu, G. Zhang, Z. Tian, C. Tian, S. Mu, X. Fan, X. Chen, and J. Wen, "Recbole 2.0: Towards a more up-to-date recommendation library," in *CIKM*. ACM, 2022, pp. 4722–4726.
[20] W. contributors, "Jaccard index," https://en.wikipedia.org/wiki/Jaccard_index, 2024, accessed: 2024-06-13.

APPENDIX

A. Example of item from the Boels' website

Fig. 3. Example of item from the Boels' website.

B. Account Data Visualization

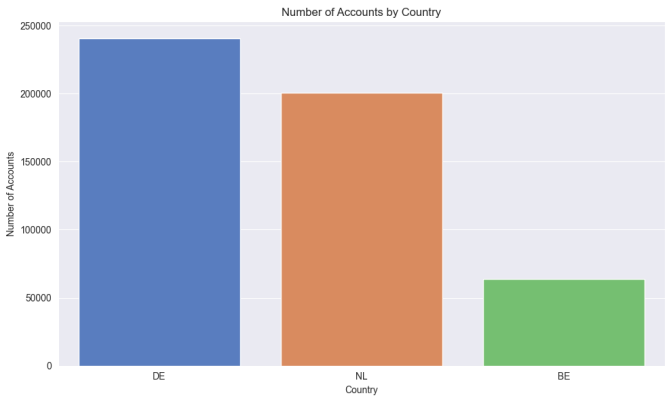

Fig. 4. Number of accounts in top 3 countries with most of rented items.

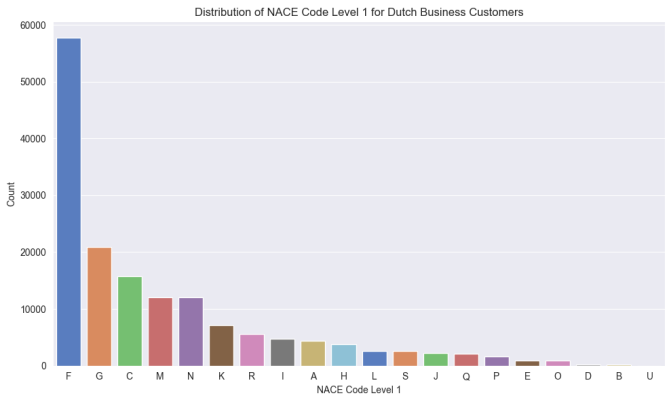

Fig. 5. Distribution of NACE Code Level 1 for Dutch Business Customers.

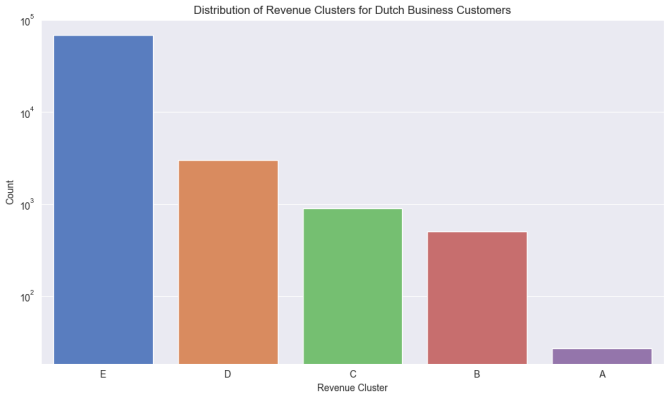

Fig. 6. Distribution of Revenue Clusters for Dutch Business Customers.

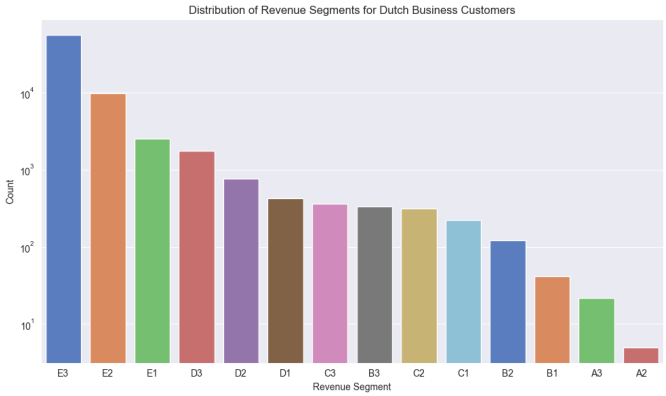

Fig. 7. Distribution of Revenue Segments for Dutch Business Customers.

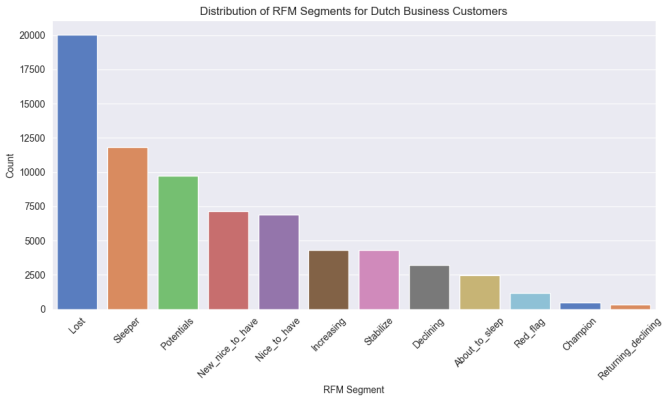

Fig. 8. Distribution of RFM Segments for Dutch Business Customers.

C. Item Data Visualization

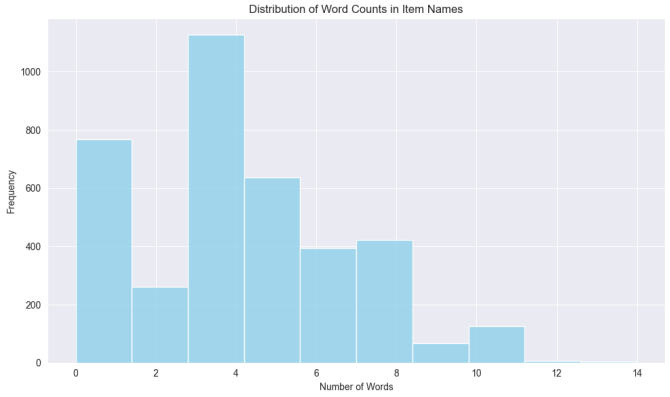

Fig. 9. Distribution of Word Counts in Item Names.

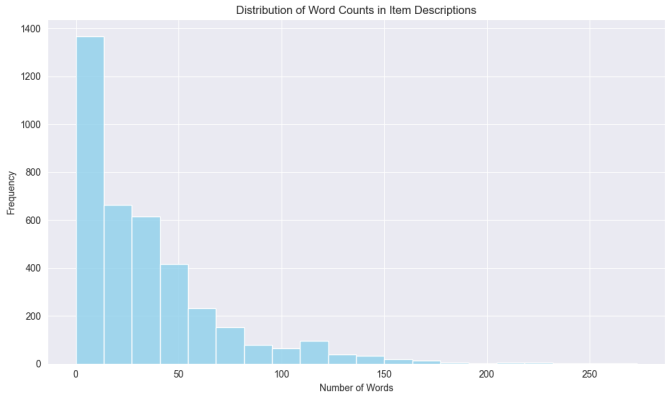

Fig. 10. Distribution of Word Counts in Item Descriptions.

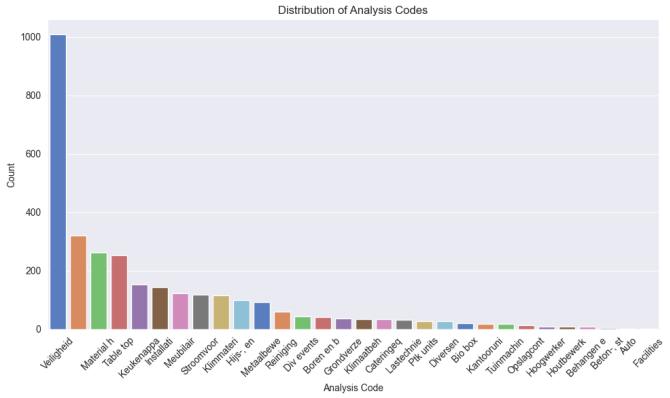

Fig. 11. Distribution of Analysis Codes.

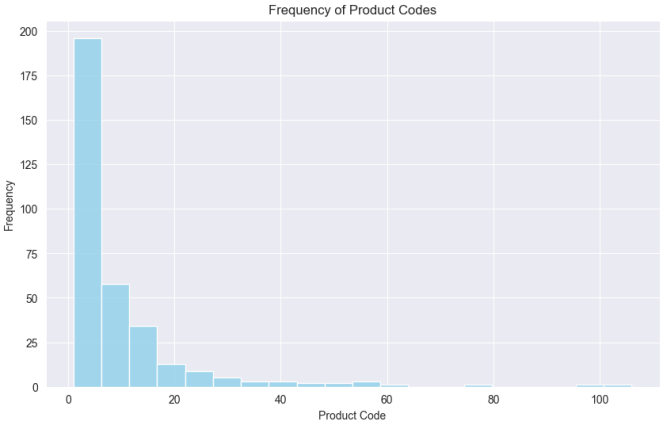

Fig. 12. Frequency of Product Codes.

D. Contract Data Visualization

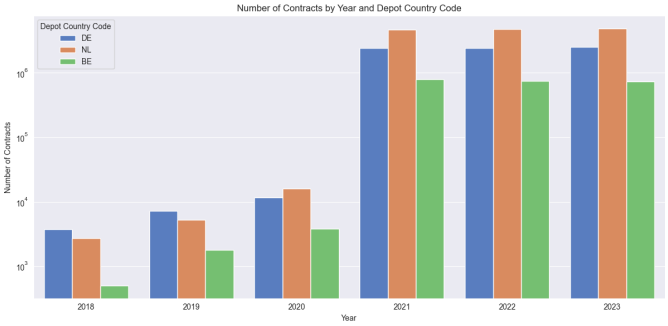

Fig. 13. Number of Rented Items by Year and Depot Country Code.

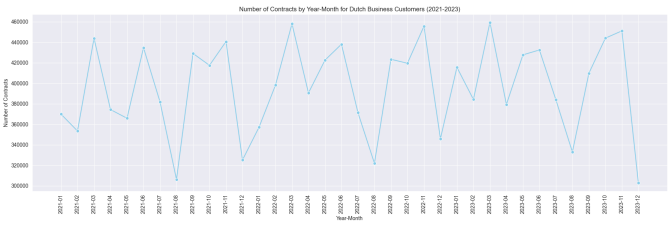

Fig. 14. Number of Contracts by Year-Month for Dutch Business Customers (2021-2023).

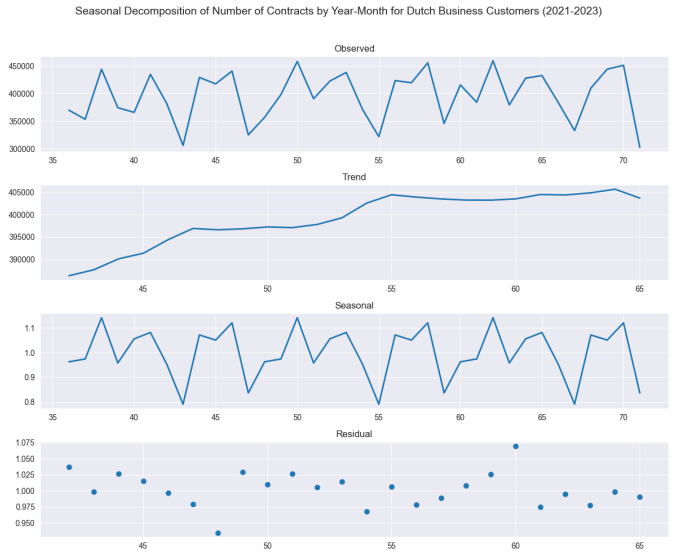

Fig. 15. Seasonal Decomposition of Number of Contracts by Year-Month for Dutch Business Customers (2021-2023).

Fig. 16. Distribution of Number of Rented Items per Dutch Business Customers (2021-2023, excluding top 5%).

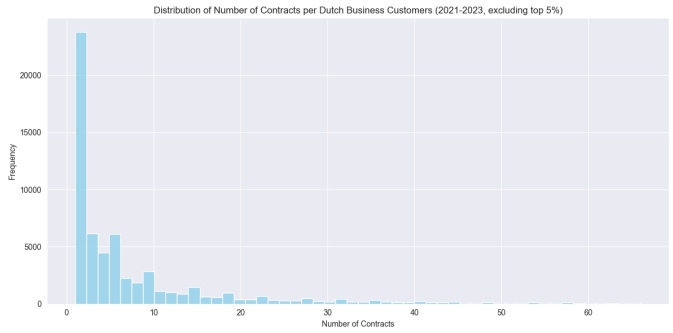

Fig. 17. Distribution of Number of Contracts per Dutch Business Customers (2021-2023, excluding top 5%).

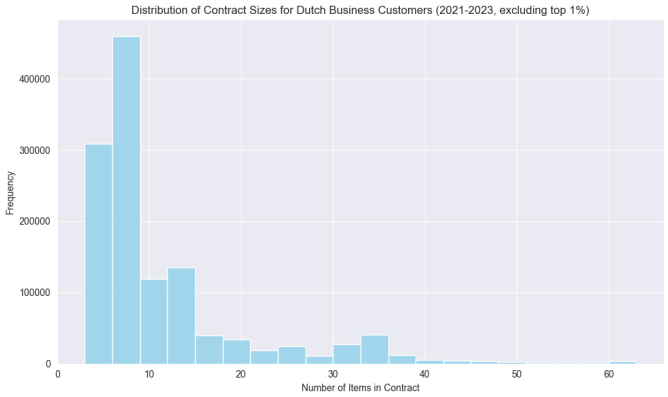

Fig. 18. Distribution of Contract Sizes for Dutch Business Customers (2021-2023, excluding top 1%).

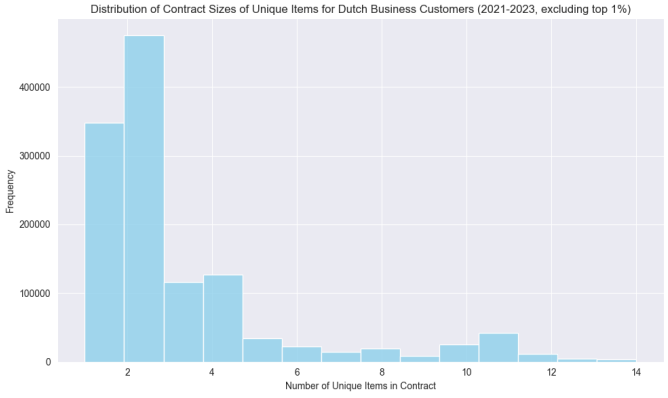

Fig. 19. Distribution of Contract Sizes of Unique Items for Dutch Business Customers (2021-2023, excluding top 1%).

E. Data Attributes Description

1) Account Data Attributes:

- *account_id:* A unique identifier assigned to each account, serving as a primary key to distinguish different customers.
- *name:* The business name associated with the customer account.
- *billing_city:* The city where the account is billed.
- *billing_country:* The country where the account is billed.
- *nace_code_lvl_1* to *nace_code_lvl_4:* A hierarchical classification of economic activities (NACE codes) at different levels, aiding in categorizing accounts based on their industry or field of operation.
- *rfm_segment:* A classification of customers into segments based on Recency, Frequency, and Monetary value (RFM analysis). This segmentation provides insight into customer loyalty and rental behavior.
- *revenue_cluster:* Clusters customers based on their revenue generation.

- *revenue_segment:* A sub-classification of the *revenue_cluster* field, segmenting accounts into groups with similar revenue characteristics.

2) Item Data Attributes:

- *item_id:* A unique identifier assigned to each item, serving as a primary key.
- *name:* The name of the product.
- *description:* A brief description of the item, providing additional information and the context in which the item is used.
- *product_code:* A code that categorizes the item into specific product groups, allowing easier identification of similar items.
- *analysis_code:* A classification code used to group similar *product_code*'s.

3) Accessory Data Attributes:

- *parent_id:* The identifier of the main product to which the accessory is linked, establishing a relationship between the accessory and its primary item.
- *accessory_id:* A unique identifier assigned to each accessory, serving as a primary key for the accessories.
- *country_code:* The country code indicating where the accessory is available, which helps in regional management and availability checks.
- *is_mandatory:* A boolean attribute indicating whether the accessory is mandatory with the main product, assisting in automatic inclusion during the rental or purchase process.

4) Contract Data Attributes:

- *contract_id:* A unique identifier assigned to each rental transaction, serving as a primary key.
- *account_id:* The identifier linking the contract to a specific customer account, enabling the analysis of account-specific rental patterns.
- *item_id:* The identifier linking the contract to a specific item, providing a direct connection between the rented product and the customer.
- *time:* The timestamp indicating the contract creation time, enabling temporal analysis.
- *depot_country_code:* A standardized code representing the country where the depot is located, providing geographical context.

F. Advanced Models Hyper-parameters

1) LightFM: The LightFM model was built using the default parameters [9], except the following ones:

- *no_components* - fine-tuned during the validation phase (dimensionality of the embedding size)
- *loss* = 'warp' (loss for implicit feedback learning-to-rank)
- *random_state* = 42

2) LightGCN: LightGCN model was built using the implementation from [18], utilizing the same default parameters:

- *lambda* = 1e-6 (regularization coefficient)
- *batch_size* = 1024 (number of samples per batch)
- *num_layers* = 4 (number of graph convolutional layers)
- *dim_h* - fine-tuned during the validation phase (dimensionality of the hidden layers' embeddings)
- *loss* = 'bpr' (pairwise ranking loss)
- *optimizer* = 'adam' (optimization algorithm)
- *learning_rate* = 0.001 (learning rate)
- *random_state* = 42

3) GRU4Rec: Our implementation of GRU4Rec model has the following parameters, with default values used from RecBole library [19]:

- *hidden_size* - fine-tuned during the validation phase (dimensionality of the hidden state in GRU layers)
- *num_layers* = 1 (number of stacked GRU layers)
- *dropout* = 0.0 (dropout rate between GRU layers)
- *loss* = 'cross entropy loss' (loss function)
- *optimizer* = 'adam' (optimization algorithm)
- *learning_rate* = 0.01 (learning rate)

4) SASRec: Our implementation of SASRec model has the following parameters, with default values used from RecBole library [19]:

- *hidden_size* - fine-tuned during the validation phase (dimensionality of embeddings and hidden states)
- *num_heads* = 2 (number of attention heads in transformer blocks)
- *num_layers* = 2 (number of transformer encoder layers)
- *dropout* = 0.5 (dropout rate for regularization)
- *loss* = 'cross entropy loss' (loss function)
- *optimizer* = 'adam' (optimization algorithm)
- *learning_rate* = 0.001 (learning rate)

G. Expanding Window Validation

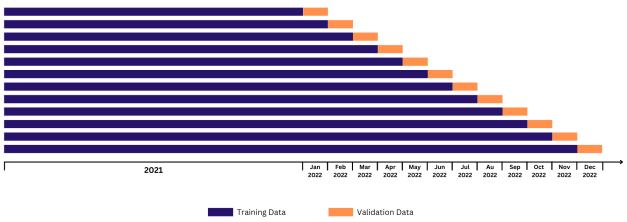

Fig. 20. Visual explanation of the implemented expanding window validation strategy.

H. Performance of the Models on the Training Data

TABLE XI
INFLUENCE OF DIFFERENT N_COMPONENTS AND EPOCHS ON MAP@5, RECALL@5, AND HIT@5 FOR LIGHTFM STANDARD

Epochs	MAP@5			Recall@5			Hit@5		
	10	30	50	10	30	50	10	30	50
5	0.1560	0.1935	0.1969	0.2615	0.3440	0.3532	0.2764	0.3609	0.3710
10	0.1648	0.1965	0.1901	0.2845	0.3595	0.3757	0.2982	0.3774	0.3960
15	0.1649	0.1885	0.1859	0.2817	0.3618	0.3732	0.2971	0.3795	0.3907
20	0.1677	0.1875	0.1806	0.2894	0.3618	0.3738	0.3044	0.3793	0.3911
25	0.1567	0.1877	0.1808	0.2776	0.3709	0.3731	0.2927	0.3866	0.3909
30	0.1597	0.1758	0.1728	0.2852	0.3680	0.3801	0.2987	0.3846	0.3973
35	0.1584	0.1763	0.1701	0.2859	0.3650	0.3714	0.3017	0.3812	0.3899
40	0.1646	0.1691	0.1689	0.2967	0.3612	0.3779	0.3125	0.3783	0.3951
45	0.1647	0.1702	0.1705	0.2964	0.3677	0.3815	0.3118	0.3842	0.3975
50	0.1612	0.1682	0.1605	0.2896	0.3597	0.3673	0.3044	0.3746	0.3813

TABLE XII
INFLUENCE OF DIFFERENT N_COMPONENTS AND EPOCHS ON MAP@5, RECALL@5, AND HIT@5 FOR LIGHTFM HYBRID

Epochs	MAP@5			Recall@5			Hit@5		
	10	30	50	10	30	50	10	30	50
5	0.0272	0.0265	0.0258	0.0624	0.0567	0.0546	0.0272	0.0601	0.0576
10	0.0331	0.0337	0.0345	0.0780	0.0660	0.0656	0.0331	0.0699	0.0700
15	0.0414	0.0401	0.0394	0.0848	0.0760	0.0740	0.0414	0.0814	0.0793
20	0.0497	0.0434	0.0447	0.0963	0.0892	0.0952	0.0497	0.0952	0.1014
25	0.0528	0.0465	0.0464	0.1078	0.1029	0.1032	0.0528	0.1093	0.1102
30	0.0538	0.0483	0.0480	0.1141	0.1086	0.1076	0.0538	0.1154	0.1145
35	0.0548	0.0484	0.0494	0.1159	0.1089	0.1100	0.0548	0.1156	0.1169
40	0.0560	0.0511	0.0519	0.1192	0.1154	0.1175	0.0560	0.1231	0.1253
45	0.0575	0.0518	0.0537	0.1226	0.1160	0.1200	0.0575	0.1238	0.1272
50	0.0575	0.0536	0.0580	0.1224	0.1193	0.1234	0.0575	0.1270	0.1310

TABLE XIII
INFLUENCE OF DIFFERENT DIM_H AND EPOCH PARAMETERS ON MAP@5, RECALL@5, AND HIT@5 FOR LIGHTGCN

Epochs	MAP@5			Recall@5			Hit@5		
	16	32	64	16	32	64	16	32	64
100	0.0021	0.0033	0.0034	0.0050	0.0093	0.0084	0.0053	0.0096	0.0091
200	0.0045	0.0035	0.0033	0.0091	0.0094	0.0095	0.0098	0.0102	0.0095
300	0.0238	0.0162	0.0079	0.0297	0.0383	0.0322	0.0311	0.0396	0.0337
400	0.0240	0.0169	0.0083	0.0299	0.0391	0.0357	0.0313	0.0404	0.0373
500	0.0239	0.0166	0.0096	0.0299	0.0389	0.0353	0.0314	0.0401	0.0370
600	0.0238	0.0163	0.0080	0.0297	0.0384	0.0355	0.0311	0.0396	0.0371
700	0.0241	0.0162	0.0080	0.0302	0.0384	0.0355	0.0316	0.0396	0.0371
800	0.0238	0.0182	0.0093	0.0297	0.0404	0.0355	0.0311	0.0416	0.0371
900	0.0251	0.0168	0.0080	0.0311	0.0389	0.0355	0.0326	0.0402	0.0371
1000	0.0238	0.0167	0.0087	0.0297	0.0390	0.0361	0.0311	0.0402	0.0377

TABLE XIV
INFLUENCE OF DIFFERENT HIDDEN_SIZE AND EPOCH PARAMETERS ON MAP@5, RECALL@5, AND HIT@5 FOR GRU4REC

Epochs	MAP@5			Recall@5			Hit@5		
	16	32	64	16	32	64	16	32	64
5	0.3282	0.3619	0.3841	0.4518	0.4970	0.5374	0.4711	0.5182	0.5603
10	0.3498	0.3706	0.3806	0.4824	0.5127	0.5358	0.5027	0.5339	0.5584
15	0.3580	0.3728	0.3781	0.4920	0.5151	0.5331	0.5125	0.5366	0.5548
20	0.3606	0.3700	0.3777	0.4943	0.5127	0.5307	0.5147	0.5335	0.5521
25	0.3628	0.3689	0.3743	0.5000	0.5120	0.5271	0.5202	0.5324	0.5483
30	0.3639	0.3686	0.3748	0.4985	0.5117	0.5261	0.5187	0.5320	0.5468
35	0.3643	0.3678	0.3734	0.5013	0.5095	0.5256	0.5213	0.5297	0.5459
40	0.3629	0.3644	0.3735	0.5000	0.5066	0.5250	0.5209	0.5261	0.5459
45	0.3642	0.3666	0.3605	0.5027	0.5095	0.5072	0.5231	0.5296	0.5264
50	0.3632	0.3670	0.3600	0.5028	0.5094	0.5059	0.5226	0.5287	0.5251

TABLE XV
INFLUENCE OF DIFFERENT HIDDEN_SIZE AND EPOCH PARAMETERS ON MAP@5, RECALL@5, AND HIT@5 FOR SASREC

Epochs	**MAP@5**			**Recall@5**			**Hit@5**		
	16	**32**	**64**	**16**	**32**	**64**	**16**	**32**	**64**
5	0.0307	0.0353	0.0338	0.0697	0.0607	0.0532	0.0772	0.0694	0.0611
10	0.0177	0.0267	0.0145	0.0337	0.0487	0.0266	0.0403	0.0562	0.0324
15	0.0156	0.0112	0.0086	0.0276	0.0329	0.0217	0.0323	0.0391	0.0247
20	0.0146	0.0118	0.0062	0.0276	0.0329	0.0116	0.0323	0.0391	0.0124
25	0.0162	0.0086	0.0100	0.0276	0.0179	0.0266	0.0323	0.0191	0.0324
30	0.0115	0.0080	0.0042	0.0192	0.0249	0.0129	0.0204	0.0294	0.0136
35	0.0116	0.0061	0.0043	0.0204	0.0175	0.0125	0.0217	0.0185	0.0131
40	0.0114	0.0081	0.0066	0.0192	0.0202	0.0203	0.0204	0.0215	0.0231
45	0.0110	0.0095	0.0041	0.0200	0.0266	0.0129	0.0211	0.0324	0.0136
50	0.0108	0.0064	0.0078	0.0192	0.0228	0.0186	0.0204	0.0275	0.0215