CHALMERS

ROIMA Parttrap® ONE

Enhancing B2B E-Commerce with a Tailored Product Recommendation System

A Data-Driven Approach to Boosting Sales and Customer Engagement

Herman Olvik
Pontus Brylander

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY & UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Bachelor's degree project report 2024

Enhancing B2B E-Commerce with a Tailored Product Recommendation System

Herman Olvik
Pontus Brylander

CHALMERS
UNIVERSITY OF TECHNOLOGY

ROIMA

Parttrap® ONE

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

Enhancing B2B E-Commerce with a Tailored Product Recommendation System
A Data-Driven Approach to Boosting Sales and Customer Engagement
Herman Olvik and Pontus Brylander

Supervisor: Johan Treptow, Roima Intelligence
Examiner: Jonas Duregård, Computer Science and Engineering, Chalmers University of Technology

Degree project report 2024
Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden

Typeset in LaTeX
Gothenburg, Sweden 2024

Enhancing B2B E-Commerce with a Tailored Product Recommendation System

Herman Olvik
Pontus Brylander

Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

This bachelor's project report presents the development and implementation of a tailored product recommendation system to enhance the B2B e-commerce platform Parttrap ONE. Leveraging machine learning models, specifically matrix factorization, the system is designed to analyze historical order data from customer websites to generate personalized product recommendations. The project focuses on evaluating and integrating Microsoft's ML.NET technologies within a .NET environment, and assess its applicability within B2B e-commerce. Key contributions include the creation of a data preprocessing pipeline, the implementation of an autonomous model training and updating mechanism, and the evaluation of model performance using precision at K (P@K) as a primary metric. The prototype system successfully demonstrates significant results in recommendation accuracy and scalability, although certain limitations such as incomplete GDPR/CCPA compliance and the need for client feedback were identified. Future work will address these limitations and explore additional features to further enhance the system's capabilities and integration flexibility.

Keywords: Keywords: B2B e-commerce, product recommendation system, machine learning, matrix factorization, ML.NET, precision at K, AI, system architecture, Microsoft .NET

Acknowledgements

We would like to thank our supervisor Johan Treptow for providing excellent guidance in the .NET environment where our own initial knowledge was lacking. This project might have looked very different without the feedback and strategies suggested in our first meeting and continuous support in our weekly meetings.
We would also like to thank Mathias Schreiner for his time during the initial consultation of this project.

Herman Olvik and Pontus Brylander, Gothenburg, May 2024

Contents

List of Figures

List of Tables

1
Introduction

In the rapidly evolving field of digital commerce, personalization has emerged as a critical factor in securing competitive advantages. Roima Intelligence, a leader in manufacturing and supply chain software, recognizes the significance of this trend. The company has embarked on enhancing its digital commerce solution, Parttrap ONE, by integrating a machine learning-driven product recommendation system. This initiative aims to refine user experiences as well as evaluating overall client satisfaction within the industrial market.

Roima Intelligence operates at the intersection of technology and industrial efficiency, providing solutions that drive the digital transformation of manufacturing processes. With an established presence in the industry, the company's commitment to enhancing digital commerce capabilities is a strategic move to meet evolving market demands.

Despite its success, Roima Intelligence faces challenges in optimizing product discovery and engagement in its digital commerce platform Parttrap ONE. Current systems do not fully leverage the potential of advanced analytics and machine learning, resulting in missed opportunities for customer user engagement and sales. The integration of a recommendation system presents an opportunity to address these gaps and significantly improve the effectiveness of Parttrap ONE.

This project is poised to set benchmarks in the application of AI within the industrial sector's digital commerce systems. By aligning with the latest advancements in software frameworks, this research will contribute insights into the effectiveness of personalized recommendations in an industrial setting.

1.1 Purpose

The purpose of this project is to facilitate the implementation of a system in a seamless manner within the Parttrap ONE platform, tailored to improve its user experience by offering personalized product suggestions to end users. This enhancement aims to increase sales efficiency for Roima Intelligence's industrial clients by leveraging advanced machine learning technologies. The purpose also extends to supporting Roima Intelligence's general proficiency within AI and machine learning technologies. By incorporating these capabilities into the Parttrap ONE platform, the company addresses immediate business needs and strengthens its technological

expertise in the B2B e-commerce domain.

1.2 Objectives

The objective of this project is to develop a functional prototype of a recommendation system. This system leverages machine learning algorithms trained on historical order data to suggest products to users. By integrating data handling and machine learning techniques, the prototype serves as a tool for enhancing customer experience and improving sales strategies.

- **System architecture**: The system manages and analyzes multiple datasets in real-time with minimal external intervention. A dedicated long-running background task operates throughout the life of the application. This task periodically fetches new data at predefined intervals, enabling the system to refresh its recommendations autonomously and stay up-to-date with the latest order trends.
- **Database integration**: A central database is integrated into the system to store order histories and model outputs. This database ensures that all relevant information is readily available for generating recommendations.
- **Machine learning models**:
  - **Training and updates**: The machine learning models at the heart of the recommendation system are initially trained on past order history data. These models are regularly updated, so that the system evolves and improves over time.
  - **Model management**: The system includes functionality for managing machine learning models through API endpoints. These endpoints allow other applications and system administrators to update model parameters and retrain models with new data.
- **Recommendations and performance evaluation**: The recommendation engine analyzes outputs from the machine learning models to suggest products that align with user preferences and buying patterns. The effectiveness of the recommendations is continuously measured using metrics as relevant to the methodology. These metrics provide insights into how well the recommendations are meeting user needs and driving sales.
- **System optimization**: The recommendation system is designed with flexibility to adapt to different datasets, allowing it to be fine-tuned for varying business contexts and customer bases. This optimization process takes into account the unique characteristics of each dataset, such as demographic factors, to enhance the relevance and accuracy of product suggestions.
- **API and integration**: The system is equipped with an API that facilitates integration with external applications, such as the Parttrap ONE system. This

API provides documentation to support internal use and modifications, enabling adaptation and maintenance. Key API endpoints include functionality for requesting product recommendations based on current order data, managing machine learning models, and updating the system with new data or configuration settings.

1.3 Limitations

Limitations made in this project are set in order to ensure the delivery of a working prototype within the available time frame of 10 weeks. The following constraints are chosen in a way such to prioritize core functionalities and lay the groundwork for future development.

- **Strict utilization of Microsoft .NET technology** A significant constraint of this project, notably in the context of general software development and particularly in the deployment and operation of machine learning models, is Parttrap ONE's stringent adherence to Microsoft .NET technology for the entire recommendation system. This exclusive reliance on .NET technology substantially limits the integration flexibility of machine learning models and restricts the analytical capabilities available for these models, given the current development state of machine learning frameworks within the .NET ecosystem. Nevertheless, this limitation is also valued as a distinctive feature of this study, as it provides an opportunity to investigate the potentialities of underutilized technologies and to boldly test and assess technologies that are not as widely recognized as industry standards such as Python.
- **Data dependency and attributes** The effectiveness of personalized recommendations is heavily tied to the datasets that the system is based on. The application is restricted to providing recommendations based on geographical and user identity-based attributes, such as city, country, user, and user company affiliation identifiers. Any adaptation to include additional identifiers or to modify existing ones requires changes to the application source code.
- **Personalization** The data sets do not include product ratings, which limits the recommendation system to primarily rely on user behavior and purchase history for generating suggestions. This approach may result in less personalized recommendations as it cannot account for user satisfaction with the products.
- **Security** The encryption process is only partially implemented; the same vector is used for every encryption. This approach is limited to encrypting user data only, while training data and prediction data stored in the database remain unencrypted.

2

Theory and technical background

The aim of this chapter is to equip the reader with essential technical background and theoretical foundations in order to understand the content of this report. This includes an overview of important concepts and principles used in the project, including the mechanics of machine learning algorithms and the ML.NET framework, as well as technical aspects involved in developing a web API application and its database integration. By understanding the key concepts, the intention is to bridge gaps in knowledge or familiarity with the subject.

2.1 Machine learning related concepts

2.1.1 .NET and ML.NET frameworks

.NET, a Microsoft-developed ecosystem, is employed primarily for applications requiring robust, scalable frameworks. ML.NET extends .NET's capabilities into machine learning, offering tools for model creation and deployment directly in .NET environments. This approach is favored in the project for integrating machine learning capabilities into existing .NET applications without the overhead of additional Python services [6].

2.1.2 Python, Mlxtend, Pandas, and Seaborn

Python remains a staple in data science due to its extensive libraries and frameworks. The 'mlxtend' library, used for implementing the Apriori algorithm for market basket analysis, provides efficient tools for rule mining and association analysis [8]. Pandas and Seaborn facilitate data manipulation and visualization, respectively, essential for exploratory data analysis in machine learning workflows [7, 11].

2.1.3 Market basket analysis and the Apriori algorithm

Market Basket Analysis (MBA) is a technique to discover associations between items within large datasets of transactions. The Apriori algorithm, a classic algorithm used in MBA, identifies frequent item sets and generates association rules from these sets. Although implemented initially in this project, its lack of personalization led to exploring other methods more suitable for individualized recommendations [1].

2.1.4 Matrix factorization

Matrix Factorization (MF) is a powerful collaborative filtering technique used in recommendation systems. It decomposes a user-item interaction matrix into the product of two lower-dimensional matrices, capturing latent factors that represent underlying preferences and characteristics.

2.1.4.1 Definition and formula

In collaborative filtering, the goal is to predict the missing entries in a user-item interaction matrix R, where R_{ui} denotes the interaction (such as rating) between user u and item i. Matrix Factorization models this interaction as:

$$R_{ui} \approx P_u^T Q_i \tag{2.1}$$

where:

- P is a user-feature matrix with dimensions $n \times k$,
- Q is an item-feature matrix with dimensions $m \times k$,
- k is the number of latent factors.

The matrices P and Q are learned such that their product approximates the original interaction matrix R. This can be formulated as an optimization problem where the objective is to minimize the difference between the observed interactions and the predicted interactions:

$$\min_{P,Q} \sum_{(u,i)\in\kappa} (R_{ui} - P_u^T Q_i)^2 + \lambda(\|P_u\|^2 + \|Q_i\|^2) \tag{2.2}$$

where κ is the set of observed interactions, and λ is a regularization parameter to prevent overfitting.

2.1.4.2 Properties and applications

Matrix Factorization has several advantages and applications:

- **Scalability:** MF scales well with large datasets, making it suitable for large-scale recommendation systems.
- **Latent Factors:** By uncovering latent factors, MF can reveal hidden patterns in user preferences and item characteristics.
- **Sparsity Handling:** MF effectively handles sparse data, which is common in user-item interaction matrices.
- **Extensibility:** MF can be extended to incorporate additional information, such as temporal dynamics or user and item biases.

2.1.4.3 Challenges

Despite its advantages, Matrix Factorization also has some challenges:

- **Cold Start Problem:** MF requires sufficient data to learn accurate latent factors, which can be problematic for new users or items.
- **Complexity:** The optimization process can be computationally intensive, particularly for very large datasets.
- **Parameter Tuning:** Selecting the appropriate number of latent factors and regularization parameters can be non-trivial and often requires cross-validation.

Matrix Factorization has been widely adopted in industry and research, particularly following its success in the Netflix Prize competition, where it significantly improved recommendation accuracy [21].

2.1.5 Implicit and explicit feedback

The distinction between implicit and explicit feedback forms the basis for understanding user interactions within matrix factorization recommendation systems. Explicit feedback involves direct input from users regarding their preferences, such as ratings [3]. On the other hand, implicit feedback relies on user actions that indirectly reflect preferences, such as purchase histories or browsing behaviors, which are utilized in this project using historical order data from Parttrap ONE's customer websites [3].

2.1.6 One-class collaborative filtering

In the context of collaborative filtering, one-class scenarios refer to situations where only positive interactions are recorded, without explicit negative feedback. This is common in real-world applications where users' implicit feedback, such as clicks, views, or purchases, is readily available, but explicit negative feedback, such as dislikes or ratings, is not [4].

2.1.7 Hyperparameters in matrix factorization

2.1.7.1 Alpha (α)

Alpha (α) is a regularization parameter that controls the penalty for large weights. In matrix factorization, regularization helps prevent overfitting by discouraging the model from capturing noise as patterns. A higher alpha value increases regularization, leading to a simpler model, while a lower alpha value might result in overfitting [30].

2.1.7.2 Lambda (λ)

Lambda (λ) serves a similar purpose to alpha but typically applies to specific factors, such as user and item latent features. Regularization with lambda ensures these feature vectors do not grow too large, maintaining model stability and generalization. Proper tuning of lambda can mitigate overfitting risks [30].

2.1.7.3 Latent features

Latent features refer to hidden factors inferred from the user-item interaction matrix. These features represent abstract characteristics of users and items that explain the observed interactions. The number of latent features is crucial: too few might oversimplify the model, while too many could lead to overfitting. Finding the optimal number involves balancing the model's capacity to learn meaningful patterns without capturing noise [30].

2.1.7.4 Number of epochs

The number of epochs denotes complete passes through the entire training dataset during model training. Each epoch allows the model to update its parameters to better approximate the user-item interaction matrix. Training for too few epochs might result in underfitting, while too many epochs can lead to overfitting. Selecting an appropriate number of epochs is vital for balancing underfitting and overfitting [30].

2.1.8 Visualization tools: Matplotlib and heatmaps

'Matplotlib', a comprehensive library for creating static, interactive, and animated visualizations in Python, is used extensively for plotting data distributions [5]. Heatmaps, generated using Seaborn, provide a powerful visual tool to represent the density and patterns of interactions between users and items, revealing underlying structures in the data [11].

2.1.9 Mean absolute error

Mean Absolute Error (MAE) is a commonly used metric in machine learning, especially for regression tasks. MAE measures the average magnitude of errors between predicted values and actual values. Unlike other metrics such as Mean Squared Error (MSE), which squares the error terms and thus penalizes larger errors more heavily, MAE treats all errors equally, providing a straightforward interpretation of prediction accuracy.

2.1.9.1 Definition and Formula

The formula for calculating MAE is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.3}$$

where:

- n is the number of observations,
- y_i is the actual value,
- $\hat{y}_i$ is the predicted value.

MAE is particularly useful when you want a simple and interpretable measure of prediction error. It is less sensitive to outliers compared to MSE, making it a preferable choice in scenarios where outliers are present or where all errors are treated uniformly [13].

2.1.10 Precision at K

Precision at K (P@K) is a metric used for evaluating the performance of information retrieval, recommendation systems, and classification tasks, particularly in the context of ranking problems. It measures the proportion of relevant items among the top K items returned by a model.

2.1.10.1 Definition and Formula

Precision at K is defined as:

$$\text{Precision@K} = \frac{|\{\text{relevant items}\} \cap \{\text{top K items}\}|}{K} \tag{2.4}$$

where:

- The numerator is the number of relevant items in the top K results,
- The denominator K is the total number of items considered.

This metric helps in understanding how well the top recommendations or predictions match the relevant items. It is particularly useful in scenarios where users are primarily interested in the top results, such as search engines or recommendation systems.

P@K is a crucial metric in fields such as information retrieval, where it is used to measure the effectiveness of search algorithms, and in recommendation systems, where it evaluates how well the system ranks relevant items for users. [14]

2.2 Software development related concepts

2.2.1 The SOLID principles

The SOLID principles, introduced by Robert C. Martin, are foundational guidelines in software engineering aimed at improving maintainability and scalability. They encourage reduced dependencies and enhanced code readability:

- **Single Responsibility Principle (SRP)**: Each class should have only one responsibility [16].
- **Open/Closed Principle (OCP)**: Modules should be open for extension but closed for modification [17].
- **Liskov Substitution Principle (LSP)**: Objects should be replaceable with instances of their subtypes without affecting program correctness [17].

- **Interface Segregation Principle (ISP)**: Clients should not be forced to use unnecessary interfaces [17].
- **Dependency Inversion Principle (DIP)**: High-level modules should not depend on low-level modules but on abstractions [17].

2.2.2 Web API

A Web API (Application Programming Interface) is a framework that enables different software programs to communicate via the web, primarily using HTTP protocols. It is essential for developing web services and cloud applications, allowing applications to exchange data and functionalities seamlessly [15]. In machine learning systems, Web APIs facilitate the deployment of models as services, which can be accessed over the web to perform predictions, supporting scalable and integrative solutions [12].

2.2.2.1 RESTful web API

A RESTful web API adheres to the principles of Representational State Transfer (REST) as defined by Roy Fielding [15]. These APIs use standard HTTP methods such as GET, POST, PUT, and DELETE and are stateless; all necessary information for the request is contained within each client-server communication, with no session state on the server.

Unlike traditional web APIs that may maintain state on the server or use non-standard protocols, RESTful APIs are efficient, scalable, and integrate seamlessly with web technologies, making them ideal for internet-facing applications [20]. Data exchanges in RESTful APIs typically occur in JSON or XML, promoting platform independence and straightforward implementation.

2.2.3 ASP.NET Core

ASP.NET Core is a free, open-source, cross-platform framework for building modern, cloud-based, internet-connected applications. It is a complete redesign of ASP.NET, offering a leaner and more modular framework. ASP.NET Core equips developers with tools, libraries, and runtime for creating web applications, services, APIs, and real-time applications with WebSockets. It supports Windows, macOS, and Linux, and provides flexible hosting options on IIS or self-hosted processes. The framework is focuses on improved performance and efficiency for high-traffic applications [22, 23].

2.2.4 MongoDB

MongoDB is a NoSQL database designed for high performance, availability, and scalability. It uses JSON-like documents with dynamic schemas, making it ideal for applications that manage large volumes of data that frequently change. This

flexibility is particularly useful in machine learning projects for storing and processing unstructured data, enabling efficient data manipulation and complex queries necessary for training models [24, 25].

2.2.5 DI-container

A Dependency Injection (DI) container in .NET is a framework component that manages object dependencies by promoting Inversion of Control (IoC). It allows for better decoupling of components, simplifies testing by enabling the use of mock objects, and centralizes configuration management. Popular DI containers in .NET include Microsoft.Extensions.DependencyInjection, Autofac, and Unity, each providing features like managed instance lifetime, lazy instantiation, and auto-registration of dependencies [26, 27].

2.2.6 AES-256

AES-256, part of the Advanced Encryption Standard family, is a symmetric key encryption algorithm with a 256-bit key, established by the U.S. National Institute of Standards and Technology (NIST) in 2001 [18]. Its robustness makes it suitable for securing sensitive data against brute-force attacks. AES-256 operates in various modes such as CBC, CFB, and ECB, each catering to specific security needs [19].

3
Methods

3.1 Machine learning approach evaluation

To find the most effective machine learning methodology, various approaches are evaluated based on their performance. Anonymized historical order data from Parttrap ONE's customer websites is used to train models capable of providing product recommendations. This order data includes the purchasing user location, its company affiliation, and all products associated with each order. With no explicit feedback available, implicit data is utilized to infer user preferences.

Minimal viable models are developed to evaluate different machine learning concepts, aiming to produce recommendations as relevant as possible and with a justified implementation complexity. .NET solutions are preferred if they can offer recommendations comparable to more widely adopted Python frameworks and provide configurable parameters to tailor the recommendations to individual users and scenarios.

A review of existing research, including Spotify's "Logistic Matrix Factorization for Implicit Feedback Data" [28], informs the understanding of matrix factorization models for implicit feedback data.

3.1.1 Market basket analysis

The initial approach was implemented with market basket analysis using Python's 'mlxtend' library. This implementation utilizes the Apriori algorithm to identify frequent item sets and generate association rules. The system requires a single product as input to output related products, but it does not provide personalized recommendations, making it less suitable for the targeted use case. Therefore, it was decided to omit market basket analysis in favor of a more suitable solution. A significant factor influencing this decision was the Parttrap ONE team's preference for .NET technology. While the initial Python-based approach did not fully meet our use case requirements, the alignment with Parttrap's technology stack played a crucial role in shifting towards a different solution.

3.1.2 Collaborative filtering

Collaborative filtering is identified as a viable approach for providing personalized recommendations by leveraging historical user-item interactions to build tailored

models. This approach is integrated into the ML.NET framework, enabling direct implementation within the .NET backend using ML.NET's Matrix Factorization support. This method reduces the need for separate services. The implementation of collaborative filtering involves structuring historical order data to include relevant attributes for implicit feedback data, cleaning and preprocessing it to ensure consistency, and preparing it for model training and evaluation.

3.2 Exploratory data analysis

3.2.1 Data preparation and anonymization

Two different Parttrap ONE website order history datasets labeled X and Y, are stored in CSV format and contains XML data embedded in each row. Python's 'csv' and 'xml.etree.ElementTree' libraries are used for parsing. To anonymize sensitive data such as stock codes, a hash function is used, truncating the hash to five characters. Each row of the CSV file is parsed to extract XML data:

- **XML parsing:** XML content is parsed to extract 'UserID' and 'StockCode' elements.
- **Data extraction:** 'UserID' occurrences are aggregated using 'defaultdict'. All 'StockCode' fields are anonymized and then aggregated similarly.

Bar chart visualizations are created using the 'matplotlib' library for the following data:

- **UserID distribution:** UserIDs are sorted based on their respective number of orders made.
- **StockCode distribution:** Similarly, the StockCodes are sorted and visualized based on their total order count.

3.2.2 Interaction heat map

Using 'Pandas' and 'Seaborn', a heatmap is generated to visualize the interaction frequency between users and StockCodes. Logarithmic scaling and a 'viridis' colormap enhance the visualization. A nested dictionary is used to aggregate interactions between users and StockCodes. Each interaction increments the count of the respective 'UserID'-'StockCode' pair. The resulting brighter colored data points in the final heat map plot indicate more interactions between that specific pair.

3.2.3 Plotted dataset visualizations

The calculations and visualizations presented in this section draw on datasets X and Y and are crucial for understanding the unique characteristics and relative popularity of specific entities within these datasets. They help elucidate the sparsity of product interactions and highlight significant differences between the datasets. This analysis informs the selection of a wide range of hyperparameter values for

the final model training. The included plots serve as an important part of the methodology, providing intermediate results that justify the methods later employed to develop a functional prototype of the recommendation system.

Table 3.1: Dataset Summary X

Metric	**Value**
Total number of unique UserIDs	486
Total number of unique StockCodes	6214
User Interaction Statistics	
Mean	10.29
Median	4.0
StdDev	37.00
Stock Interaction Statistics	
Mean	4.83
Median	2.0
StdDev	10.36

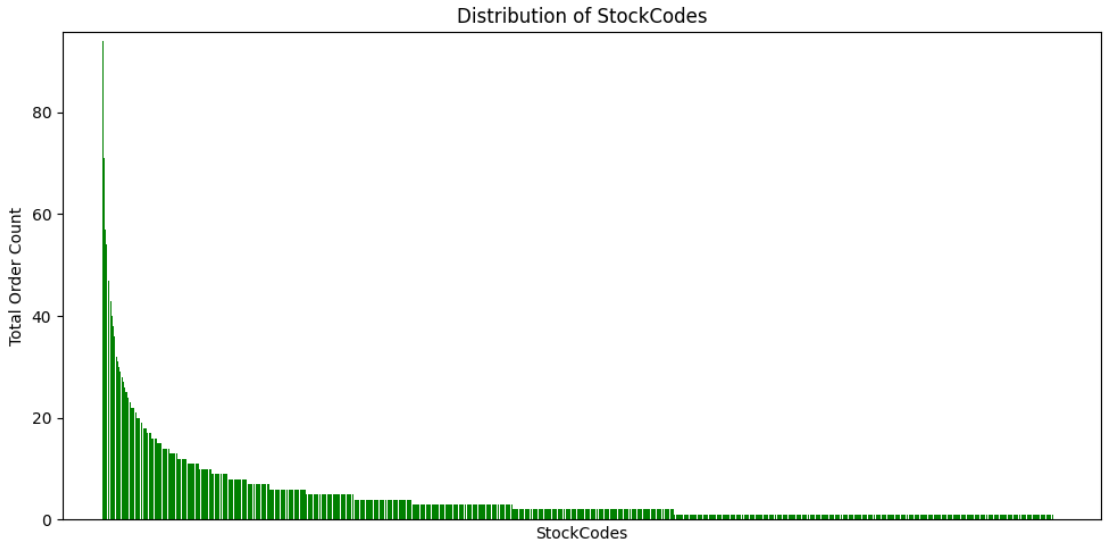

Figure 3.1: The bar chart of all products and the number of times they have been ordered in customer X's website.

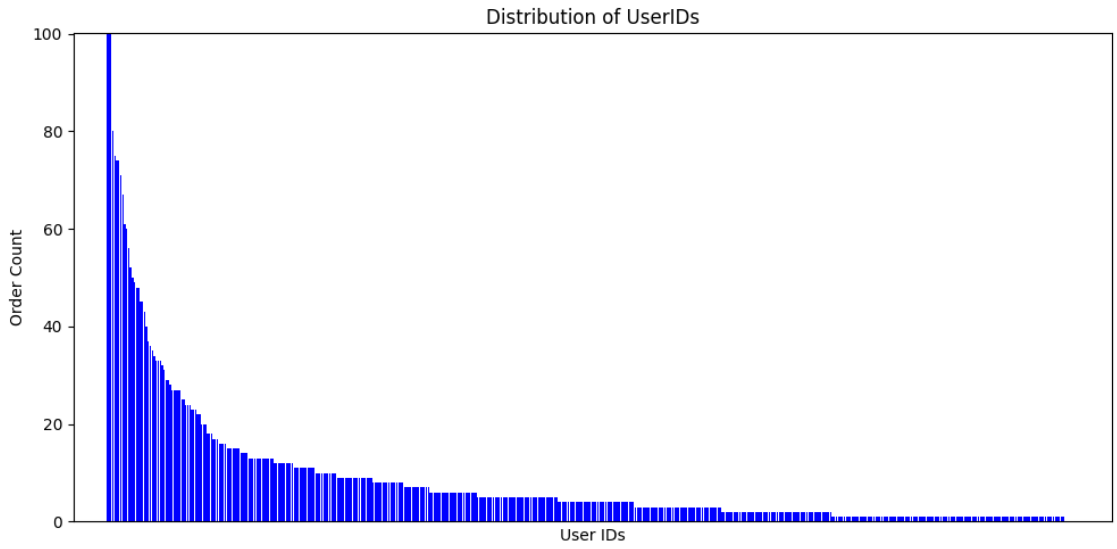

Figure 3.2: The bar chart of individual users and the number of times they have placed an order in customer X's website.

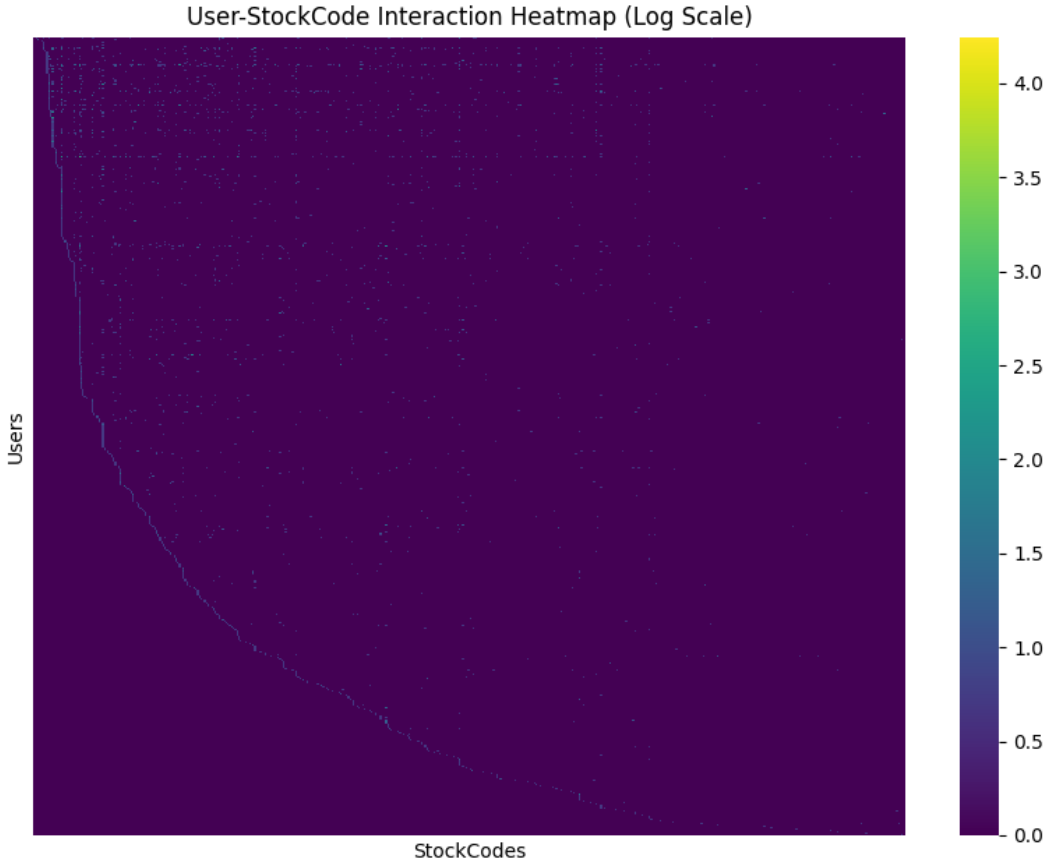

Figure 3.3: The heatmap visualizing user-product interactions on customer X's website, highlighting the dataset's significant sparsity.

3.2.3.1 Customer Y data summary

Table 3.2: Dataset Summary Y

Metric	Value
Total number of unique UserIDs	49
Total number of unique StockCodes	281
User Interaction Statistics	
Mean	10.69
Median	2.0
StdDev	32.34
Stock Interaction Statistics	
Mean	5.90
Median	1.0
StdDev	24.12

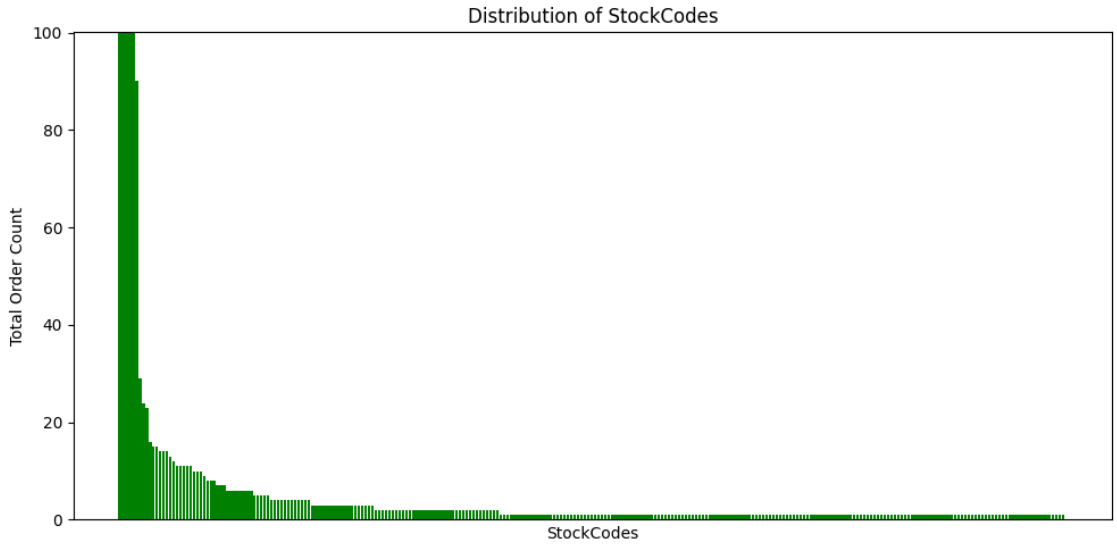

Figure 3.4: The bar chart of all products and the number of times they have been ordered in customer Y's website.

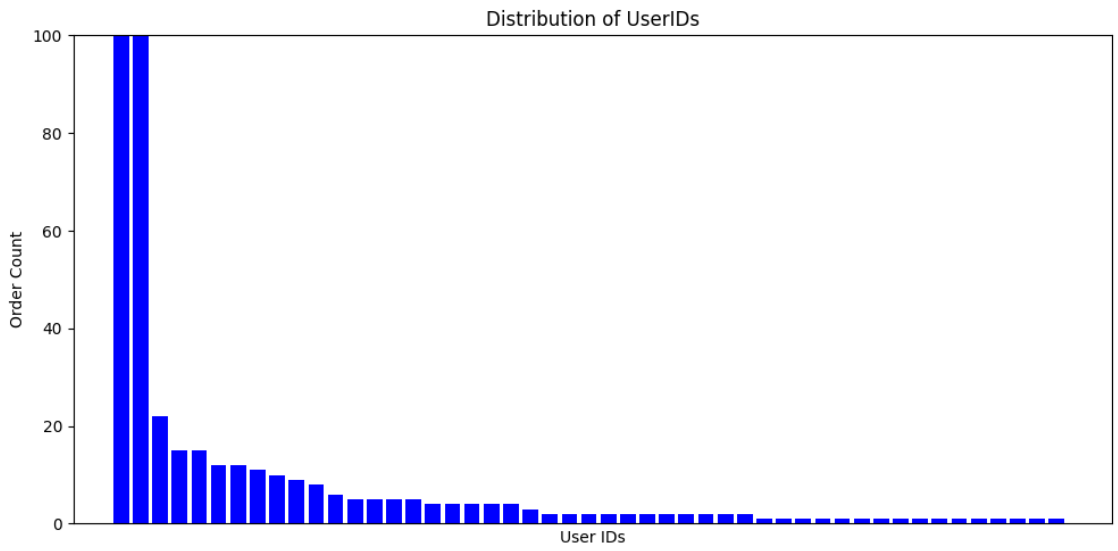

Figure 3.5: The bar chart of individual users and the number of times they have placed an order in customer Y's website.

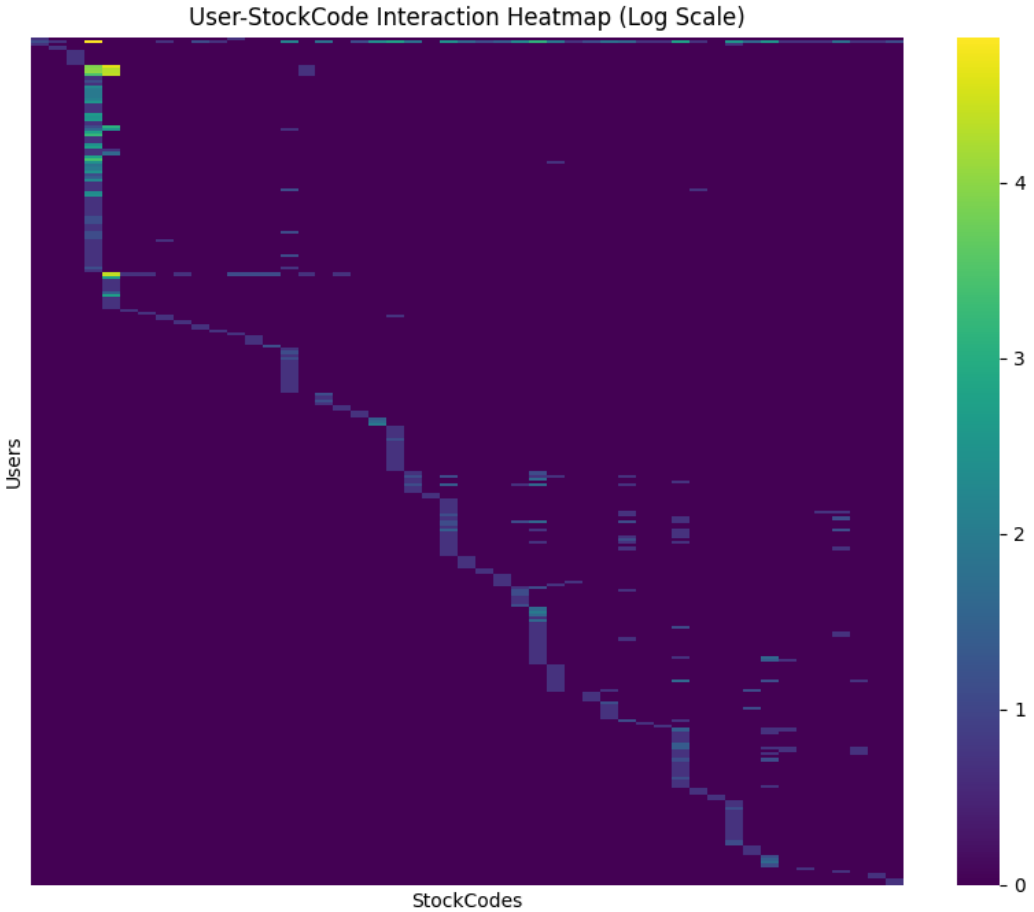

Figure 3.6: The heatmap visualizing user-product interactions on customer Y's website, highlighting the dataset's significant sparsity.

3.3 Matrix factorization task implementation

The matrix factorization task is implemented using the ML.NET framework. The process is designed to be autonomous and effective on a broad range of order history data sets, given that the exploratory analysis showed substantial differences from one customer website order history to another. The task starts by initializing groups of product interactions by various categories. These categories are as follows:

- customer company affiliation
- user id
- zip code
- country name

and they are each used to build its own matrix factorization model. The collected data is then processed to prepare it for training and evaluation phases, which finds an optimized combination of hyperparameter values for each category model. Once these final models have been trained, they are immediately used to generate prediction scores for each product and categorical entity within each category, resulting in a final data structure of predictions covering every category along with a general accuracy score for each category, as derived by an 'average precision at K'-score, whose details are covered in 3.3.3.

3.3.1 Data preprocessing

The data preprocessing step involves transforming categorical identifiers into numerical keys using the MapValueToKey transformation, which applies an index encoding. This transformation assigns unique integer values to entity identifiers (user id, customer company affiliation, etc.) and product identifiers, since only numerical data is allowed. The target variable representing the label of user interaction is also converted into a numerical format (1 for interaction, 0 for non-interaction). The processed data is split into training and testing sets in an 80/20 ratio, to facilitate both model training and evaluation.

3.3.2 Model training and parameter optimization

To account for the broad range of data set characteristics, the hyperparameter combinations found in 3.3 are evaluated for each category of training data, resulting in 3 x 4 x 4 x 3 = 144 different combinations for each category where each combination will be considered its own training cycle and evaluation step.

The matrix factorization model and hyperparameter ranges selected are driven by the dataset's characteristics: high data sparsity, implicit feedback, and user/product count imbalance.

The "SquareLossOneClass" option is used between training epochs as the loss function during each combination's training cycle, which activates one-class matrix factorization specifically designed for implicit-feedback scenarios such as this one.

Parameter	Values considered
Epochs	20, 50, 100
Alpha	0.0001, 0.001, 0.01, 0.1
Lambda	0.001, 0.01, 0.05, 0.1
Latent features	3, 5, 10

Table 3.3: Value combinations considered for the hyperparameters of the matrix factorization model.

3.3.3 Evaluation metrics

After each training cycle, the model is evaluated using a set of metrics as outlined below. The results of each model evaluation are stored in a list, where each entry represents the results of a unique model with its specific combination of hyperparameter values, along with its MAE and average P@K evaluation metrics.

- **Mean absolute error:** The mean absolute error for each combination is obtained using ML.NET's prebuilt Regression.Evaluate method.
- **Average precision at K:** A custom algorithm implementation serve to calculate an average precision at K-score for a given product interaction category (user id, customer company affiliation, etc.), based on precision at K-scores for each entity (such as a single user or a single zipcode) within that category. The pseudocode as derived from the actual implementation can be outlined as follows in 3.8, where a "prediction group" corresponds to a specific category. Pseudocode for the specific "calculate precision at k"-step for a single prediction group is outlined in 3.7.

3.3.4 Model selection based on P@K

The model whose average P@K is the highest is selected as the best model to be used for each category. A K-value of 10 is selected for every calculation, as this is an estimate made in alignment with Roima Intelligence as a rough approximation of the maximum number of product recommendations to be made during a single session in a scenario where the final system is used.

3.3.5 Generating predictions

Product predictions are generated for each category using their respective best models and stored in the application database. Only the top 50 products for each categorical entity is stored to minimize the total size of the stored data.

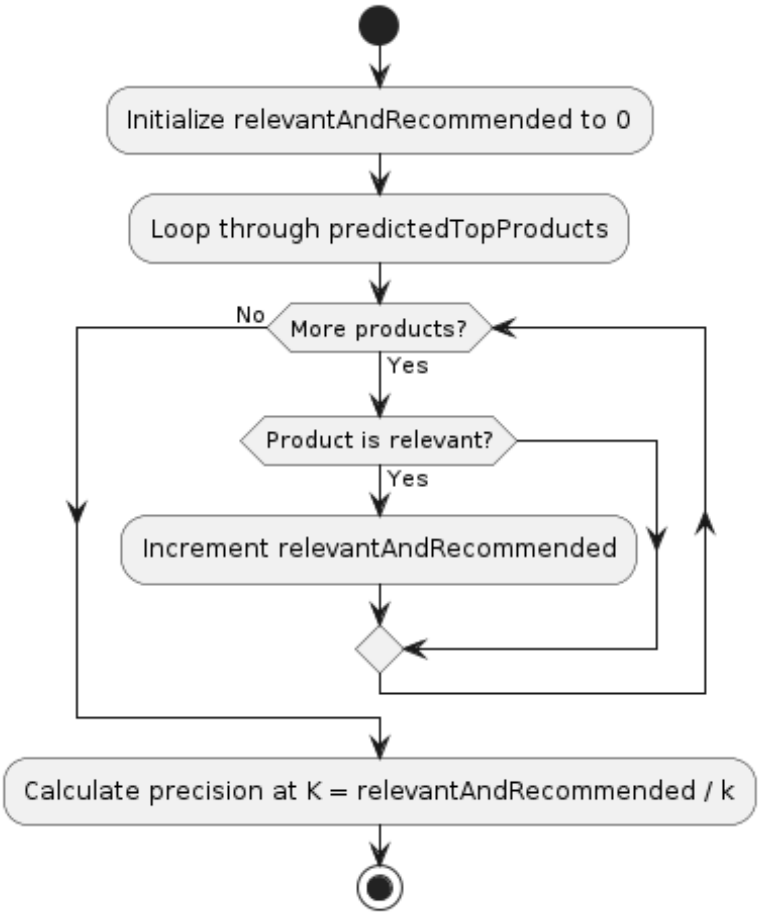

Figure 3.7: Pseudocode illustration of the custom implementation of precision at K calculation algorithm.

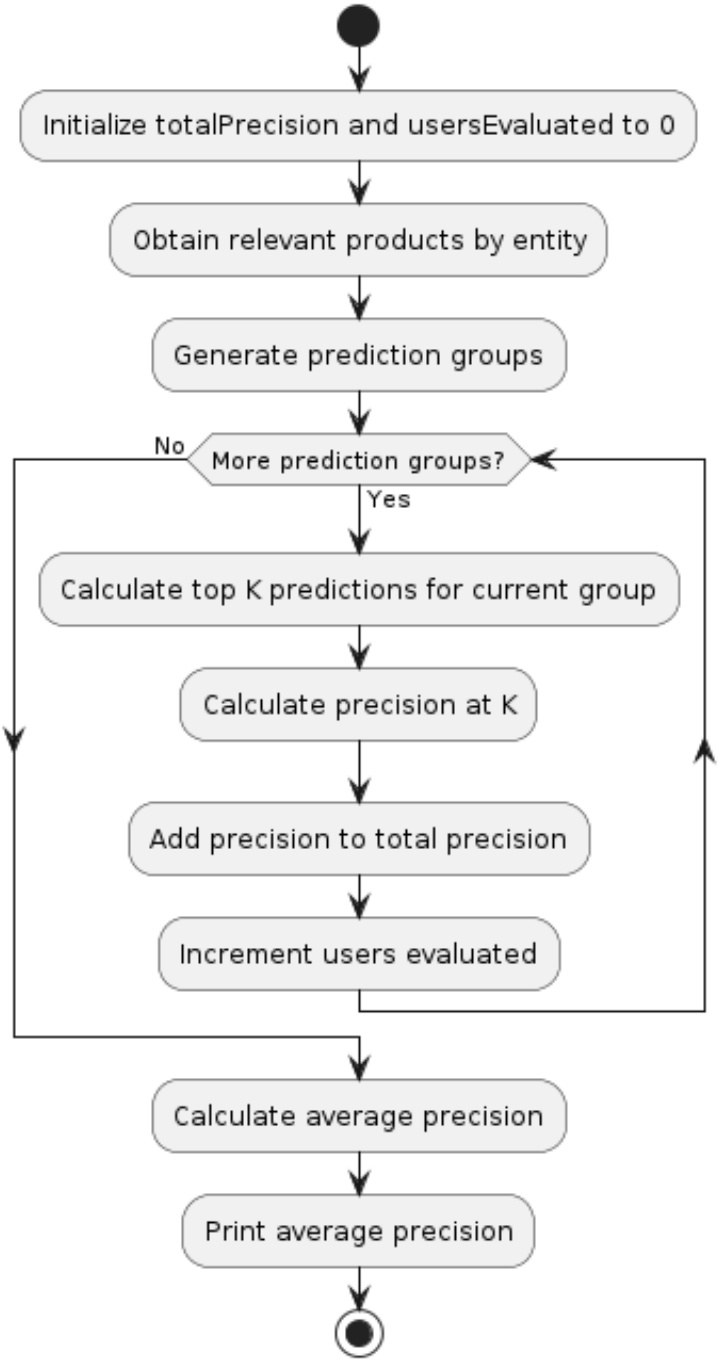

Figure 3.8: Pseudocode illustration of the custom implementation of average precision at K calculation algorithm.

3.4 General architecture of the application

3.4.1 Polling

A polling mechanism is implemented as a fundamental component of the architecture to support autonomous updates of product recommendations based on new data. The mechanism functions as the driver of the matrix factorization task, allowing the system to continuously adapt to new data without manual intervention.

3.4.2 General implementation overview and data flows

The polling mechanism can be configured to retrieve its data through an API request to another service, or directly from a database using a connection string, as well as the interval of each polling event. Therefore, a unique mechanism for each user is provided to accommodate these distinct preferences. Once data is successfully fetched, a sorting mechanism structures the data into product interaction objects used by the matrix factorization task. Following a successful polling and sorting operation, an update within the training module is automatically triggered to refresh the recommendations based on the new data. This overview is illustrated in 3.9.

3.4.3 Initial approach

Multiple implementations of the polling mechanism are tested. Initially, transient style classes are used, with each polling instance represented by a new class registered in the DI-Container. These classes autonomously manages polling, data sorting, and the timing of repeated polling events. However, managing the creation and destruction of these classes through a singleton background task, using the .NET framework's IHostedService for long-running tasks, is deemed too complex. This complexity motivates the exploration of alternative approaches.

3.4.4 The resulting polling mechanism

The initial polling approach is refined by enhancing the role of a manager class, which now not only creates and destroys polling instances but also adjusts their intervals and centralizes the core functionalities. The polling sequences are streamlined into a simpler data structure, as depicted in 3.10. These details are stored in a concurrent dictionary alongside a cancellation token, which facilitates management of processes during updates or deletions.

Furthermore, the polling function itself is transformed into an asynchronous operation. This allows it to manage multiple polling actions concurrently without significant delays. To automate the polling mechanism, each polling sequence object is equipped with a timer. This timer is linked to a callback function that triggers the polling function, thus ensuring efficient data polling across the system.

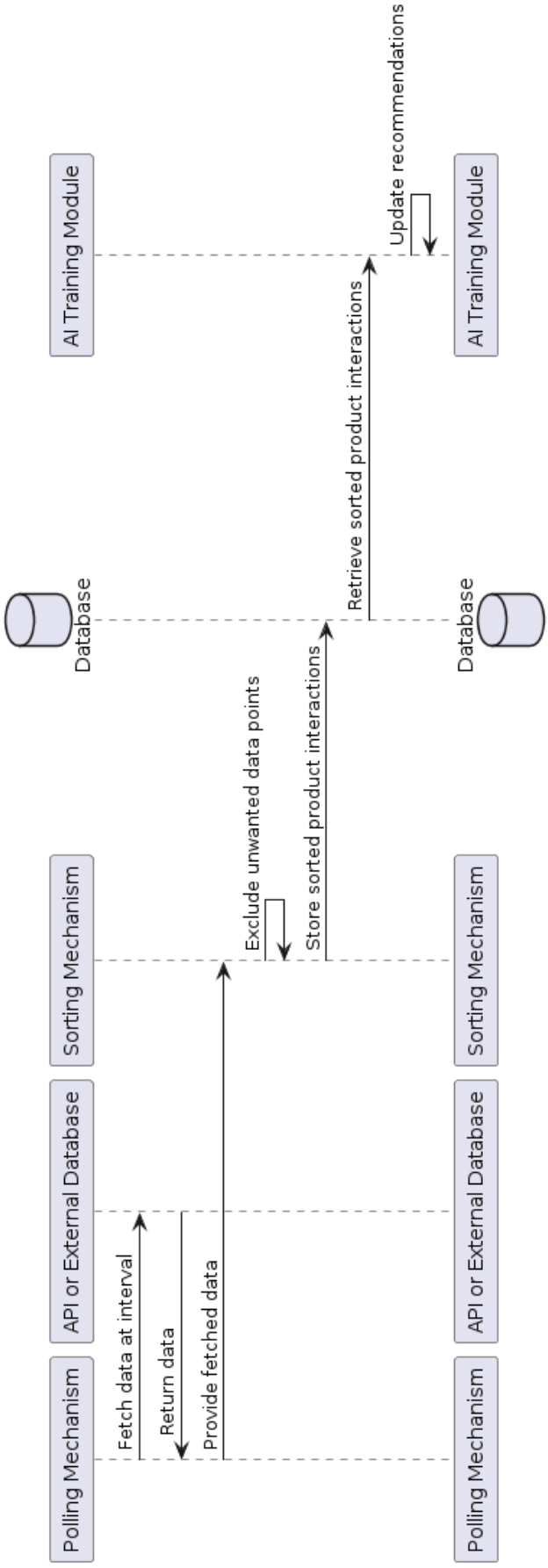

Figure 3.9: Diagram of the general flow of data within the application.

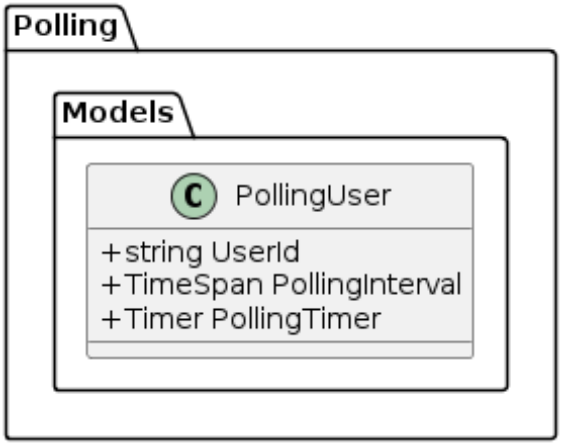

Figure 3.10: A representation of the object used in a polling sequence for one user.

3.4.5 DI-container

A DI-container is served by the .NET framework to implement dependency injection. Three main types of registrations are utilized: Singleton, Transient and Scoped. Singleton is chosen for long lived tasks that manage resources which should not be duplicated, such as polling related classes involved in background tasks throughout the applications life cycle. Transient registration is chosen for classes where each instance corresponds to the training of a dataset. Lastly, scoped registrations are mainly used for classes that handle database access, ensuring that the connection is appropriately managed within the scope of a request.

3.4.6 Data structures

Data structures are utilized to store data across the entire application. These data structures assist with type checking during the processing of an API request, which is automatically handled by utilizing these objects as templates, thus helping maintaining the consistency and correctness of data throughout the life-cycle of the application.

Adopting MongoDB's schemaless nature, the predefined structures are significantly beneficial when storing data. It allows the application to store these structured objects directly, which simplifies handling of data and reduces the need for data transformation otherwise required by relational databases.

- **The user structure:** As shown in 3.11, this structure is used to store user data, allowing for a customized polling of data based on the parameters as outlined. "Excluded Stock Codes" can be used to omit certain items from the polling process, thus excluding them from training. For clarification, a 'user' here is equivalent to an entire e-commerce website belonging to a customer with their unique order history data, and not a user within that website.

- **Structures used in the training process:** Illustrated in 3.12, represent the objects that are used to supply the training process with data to train on. The structure houses sorted data processed by the polling mechanism. A single instance of the TrainingDataModel represents a purchase order of an individual item along with information about the purchaser. These are grouped in a list within an instance of a TrainingDataModelGroup object. The UserId within the TrainingDataModelGroup represents a single user as outlined in 3.11.
  Following an initiated training process, a status string is stored within the database utilizing the object shown in 3.13. The status is then continuously updated to provide information on the current status as outlined in the figure. It includes a unique token that serves as the identifier for the status object which is used when requesting the status of a certain training process.

- **The recommendation structure:** As shown in 3.14, the 'RecommendationModel' consists of nested 'PredictionModel' structures containing precalculated predictions formed after training. Each 'PredictionModel' includes a precision score that reflects the average precision at K from the training, indicating the utility of the predictions in each category as recommendations.

  Within each 'PredictionModel', a 'PredictionGroup' contains specific entities, for instance, in the category of countries, the 'entityId' might be "Germany" or "Sweden". This grouping allows for targeted predictions based on the category distinctions. At the base layer, the 'ProductPrediction' specifies individual items to be recommended. Each product prediction includes a score indicating the likelihood of the item being bought, accompanied by a 'StockCode' that serves as a unique identifier for the item.

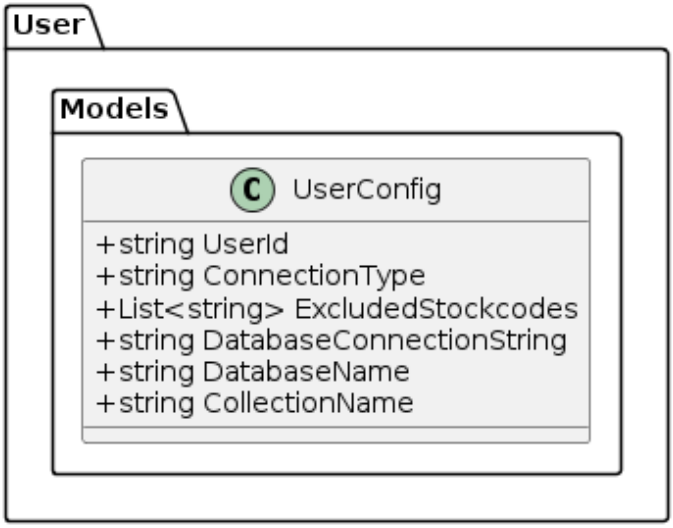

Figure 3.11: UML diagram of the structure used to store user specific data.

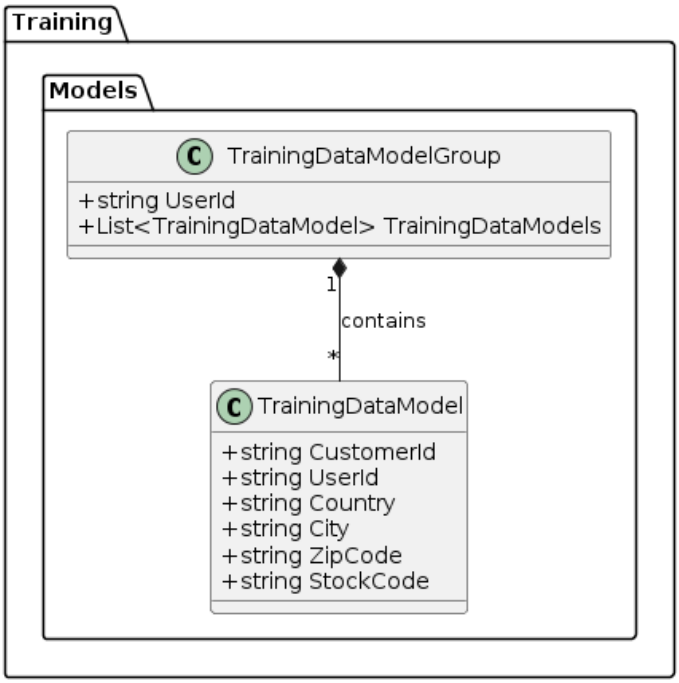

Figure 3.12: Diagram of the data structures that are utilized for storing training data used in the training process.

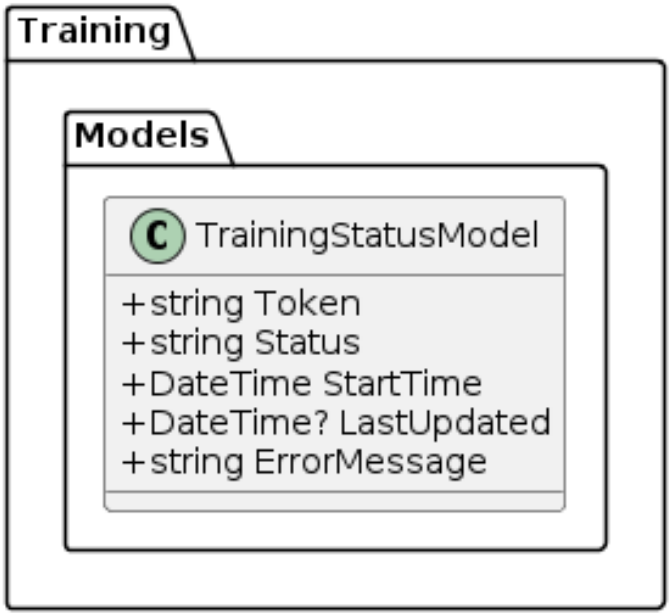

Figure 3.13: Diagram of the object used for handling the status of a single training process

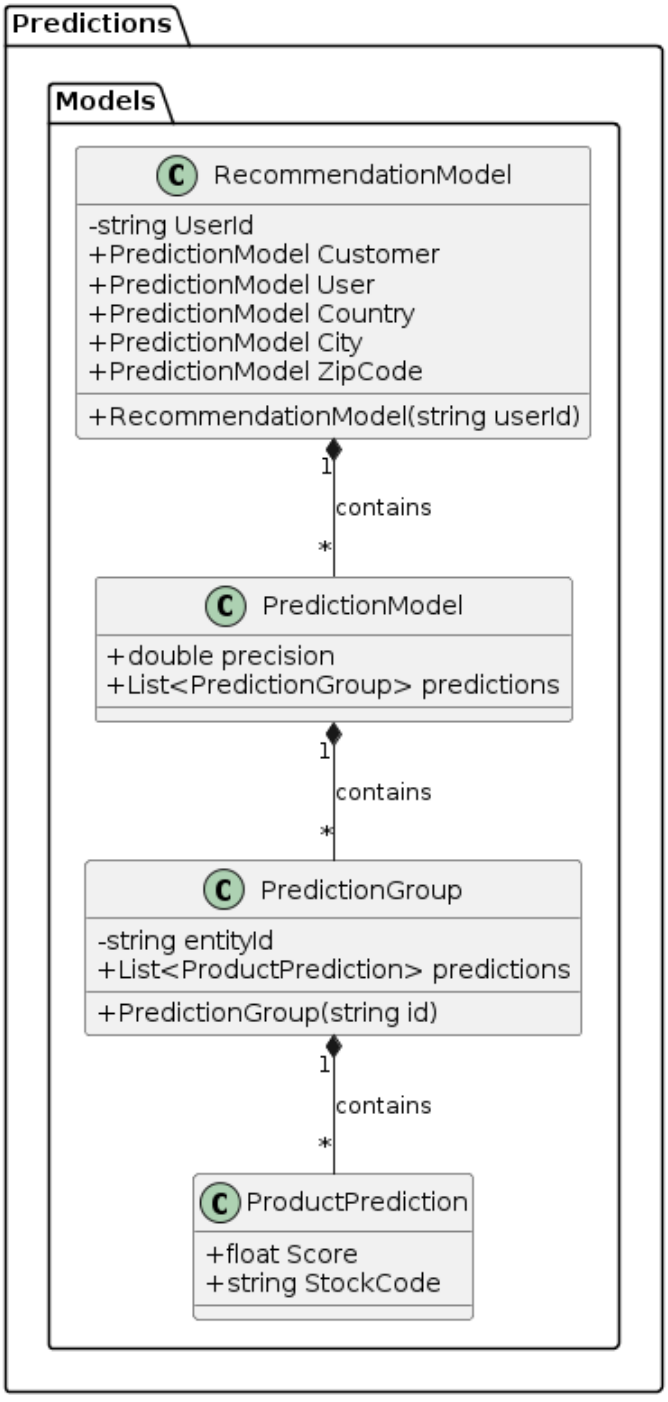

Figure 3.14: Representation of the nested structure of pre-calculated recommendation data, resulting from a training session.

3.5 REST API

A RESTful web API is implemented in the application using ASP.NET Core. The API is divided into four controllers, each dedicated to separate categories: UserController for user management, PollingController for managing the autonomous polling of data, TrainingController for managing matrix factorization model training sessions, and PredictionsController for retrieving recommendations based on the given category. This design choice is made to clearly separate responsibilities. It further prevents cross-contamination between dependencies, safeguarding, for example, user management to not interfere with polling functionalities.

A detailed list of the API and its parameters is shown in the **API Documentation Appendix**

3.5.1 User management and related data

The application handles all its users in the form of the user objects shown in 3.11, identified by an unique 'UserId' field, corresponding to the name of the company for example. A 'UserController' class handles user-specific requests such as adding, removing and updating users in the system. Adding a new user to the system involves securing sensitive data, such as API queries containing API keys and database connection strings. This data is encrypted using AES-256 and stored in the database. When removing a user from the application, the data is deleted from the database. Additionally, if the user is actively running any operations during the deletion, these operations are terminated by utilizing cancellation tokens.

3.5.2 Polling

Polling collects specific order data, sorted and used to train a recommendation model, based on user preferences. Polls are scheduled at user-specified intervals, defaulting to daily if unspecified, and can be adjusted via the REST API. Upon a StartPolling request, the system verifies user existence in the database; polling does not start if the user is absent. Polling begins immediately after initiation and continues at the set interval until manually stopped, the user is deleted, or an error occurs. Overlapping polls are managed using cancellation tokens to ensure safe termination. Polling intervals for ongoing sequences can be updated, with a new interval taking effect immediately after an initial poll.

3.5.3 Training

Training recommendation data is typically automated within the application but may require manual intervention for testing and monitoring purposes. The training process is linked to the polling mechanism, automatically queuing new data for training as it becomes available. If manual training initiation is necessary, a command can be used to start a session with the most recently polled data. Each training session generates a unique token, which is provided upon initiating a forced

training. This token allows real-time monitoring of the training's progress.

3.5.4 Retrieving predictions

The API also provides access to the final product recommendations through the PredictionsController. As the predicted products are pre-calculated from the training task, no time is wasted on calculations upon retrieval. The endpoints provided by this controller offers product recommendations according to their different categories; user id, company affiliation, country name, and ZIP code.

4
Results

This section of the report delves into the outcomes and results of the project, providing a detailed analysis of both the successes achieved and the compromises made throughout the project's execution. It outlines the effectiveness of various strategies and decisions, assessing their impact on the project's results. Additionally, the chapter explores different metrics and measures used to evaluate these results, offering insights into how well the project met its objectives and where adjustments were necessary.

4.1 Recommendation accuracy measures

The effectiveness of the recommendations for the evaluation customer websites references as 'X' and 'Y' were measured using precision at K (P@K), and provided insights into how many of the recommended products based on an early set of order data for a specific entity were later bought by the same entity according to later order data. Following the autonomous polling and training sequence, a K-value of 10 was strictly used for determining the best set of hyperparameters for a given model, as outlined in 3.

Figures 4.1 and 4.2 showcase how those P@K values were affected by altering three of the total four hyperparameters; alpha, lambda, and the number of latent features, excluding the number of training epochs. These results were all provided specifically based on recommendations associated with the customer company affiliation identifier, which was one of the categories of product interactions processed by the prototype. This category was one of special interest by Roima Intelligence.

4.1.1 Model P@K highscores

The table 4.1 showcase the calculated accuracies from the different model categories for both evaluated data sets X and Y.

4.2 Summary of the resulting prototype system

The project successfully reached its primary goal of creating a functional, autonomous recommendation system prototype. When comparing all outlined prototype-related objectives in 1.2 with 3, it is evident that some aspects were compromised. Below

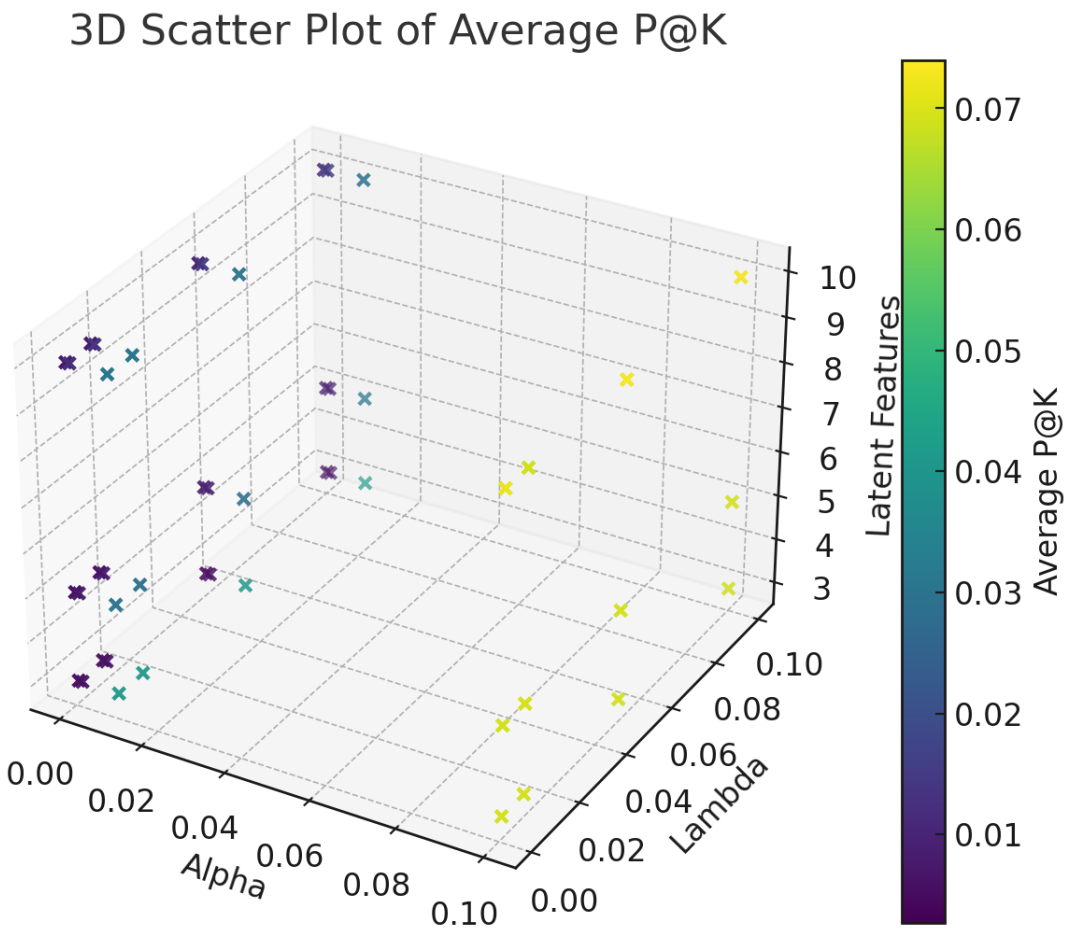

Figure 4.1: A scatter plot illustrating average P@K scores for customer X's customer company affiliation models by alpha, lambda, and the number of latent features.

Customer	User company aff.	UserID	Country	ZipCode
X	0.0779	0.0733	0.6167	0.0408
Y	0.1588	0.1579	0.4500	0.1466

Table 4.1: Accuracy high scores from the different model categories for both evaluated customer data sets X and Y.

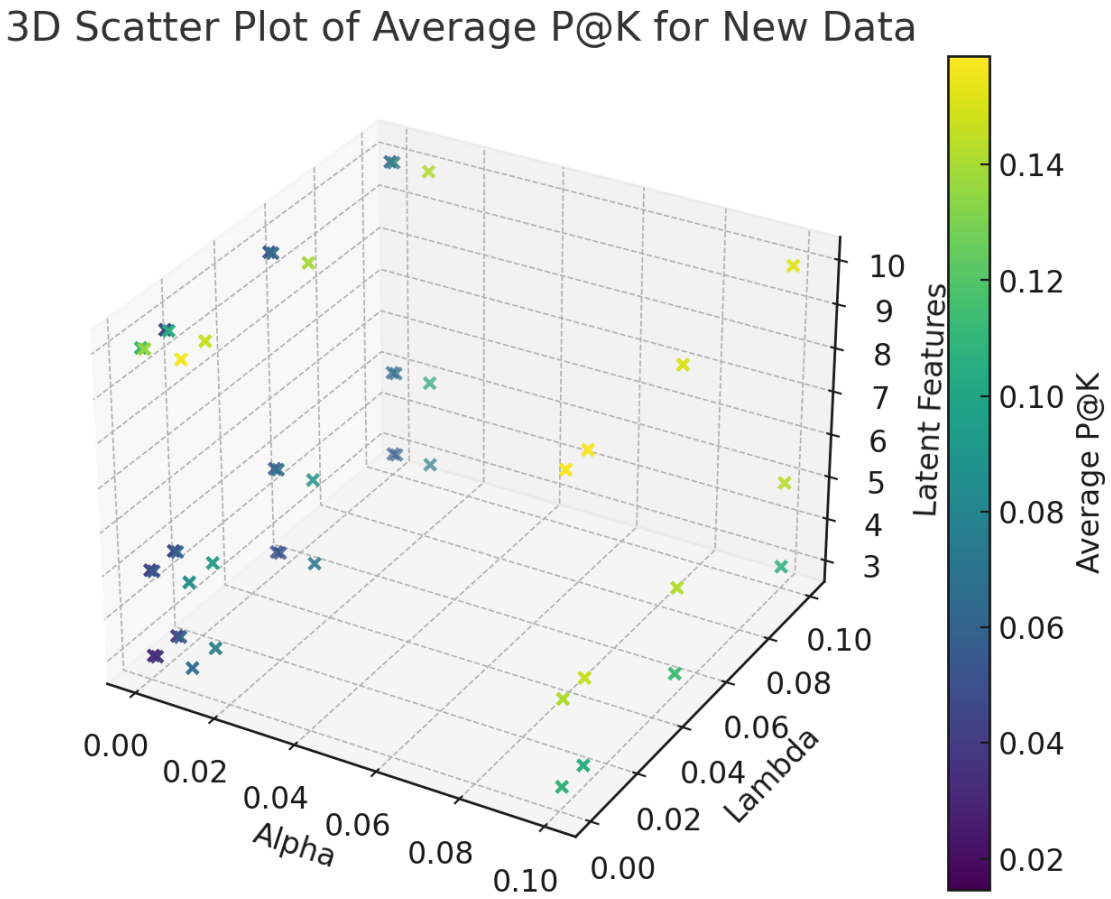

Figure 4.2: A scatter plot illustrating average P@K scores for customer Y's customer company affiliation models by alpha, lambda, and the number of latent features.

is a brief, concrete overview of what objectives in regards to the prototype were successes and not, and what the compromises were.

4.2.1 Successes

- **System architecture**: All objectives were met as outlined. The system architecture was designed to support modularity and scalability by following the SOLID principles throughout the planning and development.
- **Database integration**: A MongoDB instance was successfully integrated to handle the application's data storage needs. By employing the principle of dependency inversion, the dependencies on MongoDB were abstracted from the main application code. This approach not only ensures a clean separation of concerns but also allows for easy interchangeability with other database systems in future implementations, should the need arise.
- **Utilization of machine learning models**: Matrix factorization models were successfully implemented using ML.NET, enabling the generation of accurate recommendations even with sparse training data.
- **System optimization**: The final prototype supports the autonomous evaluation of hyperparameter grids, ensuring a more robust system capable of handling various data types and sizes.
- **API and related integration aspects**: Endpoints were designed following REST principles, providing a consistent and predictable interface for external applications.

4.2.2 Compromises

- **Performance evaluation of an actual use case**: the prototype does not offer any way of measuring how well the recommendations are driving sales in an actual use case, which was stated as an objective.
- **Complete support for GDPR and CCPA appliance**: apart from the data anonymization aspects outlined in 3, no additional specific consideration for GDPR or CCPA was taken into account for the final prototype.

4.3 Client research and feedback

Regrettably, no client research study was undertaken, nor were any clients engaged in offering feedback during the project timeline as sufficient information to conduct the prototype implementation was provided by the supervisor as well as from other employees at Roima Intelligence. Nonetheless, plans are underway to conduct such a study in the imminent future, as the project advances and specific details are being finalized in preparation for further proceedings.

5
Discussion

In this chapter, we focus on a detailed analysis of the results from the system implementation, particularly emphasizing their alignment with the initial purpose set for the project. We explore whether the system effectively addressed the issues it was designed to solve, and assess areas where it may have deviated from expected outcomes. This alignment analysis is meant to enhance understanding the system's effectiveness and guiding future research, enhancements or adjustments.

5.1 Machine learning implications

5.1.1 Market basket analysis approach

During internal discussions with the supervisor at Roima Intelligence, the market basket analysis approach, while recognized as viable, was ultimately not selected for the final prototype. Primarily, the method was not considered sufficiently personalized to meet the project's objectives. Market basket analysis only provides global associations between products rather than personalized recommendations tailored to individual users' needs [45]. Consequently, while this system can suggest products based on previous purchases, it fails to adjust its recommendations according to unique user profiles or current customer behavior. This is particularly restrictive in scenarios where the platform aims to offer recommendations without requiring users to explicitly search for products.

Furthermore, relying solely on input stockcodes raises additional challenges regarding which stockcodes should be used to generate relevant recommendations for an individual user. This could lead to a situation where recommendations are made in relation to products the user is already familiar with, rather than introducing new and more relevant options [46].

5.1.2 Finding the right accuracy metric

Initially, upon implementing a matrix factorization task, the Mean Absolute Error (MAE) was employed as the evaluation metric, sourced from the ML.NET library. This metric was assumed to accurately reflect model performance on the test set following training on the training dataset [29]. MAE was utilized to determine the optimal set of model hyperparameters. However, further analysis revealed that achieving a favorable MAE score could be misleadingly simple, as the model could improve its MAE by predominantly predicting non-interactions, regardless of actual

user-product interactions [30].

The flaw in relying solely on MAE is its equal treatment of interaction and non-interaction predictions, which does not account for the sparse nature of the datasets, where most users have not interacted with the majority of products. This was highlighted during our exploratory data analysis, showing that assuming non-interactions would statistically confirm the model's predictions in over 99% of cases [31].

To address this, we explored alternative metrics that more effectively capture the quality of the model's predictions. Precision and recall were identified as suitable metrics for evaluating model performance in scenarios characterized by data sparsity [32]. These metrics emphasize the model's ability to predict actual interactions, which is more aligned with the objectives of our study. Implementing these metrics required binarizing the interaction data, which involves setting a threshold to distinguish between interactions and non-interactions, adding a layer of complexity due to the diverse characteristics of the dataset [33].

Further research led to the discovery of Precision at K (P@K) as an even more robust metric. P@K evaluates the precision of the top K recommendations made by the model, providing a reliable measure of model performance irrespective of the prediction score's distribution and magnitude [34]. This metric is particularly advantageous for its adaptability to the varying scales of user interactions and its focus on the relevancy of the top-recommended products.

5.1.3 Implications of achieved P@K scores

Considering the P@K score achieved by customer Y's UserID model, an average of 0.1579 indicates that approximately 16% of the recommended products are actually purchased. This level of precision, while modest, is noteworthy as it suggests that nearly two out of every ten recommendations result in a purchase, underlining the model's capability to capture specific buying behaviors on the customer's website [35].

However, this does not necessarily reflect the model's impact on increasing product sales, which remains a limitation of this evaluation technique. The focus is rather on the model's ability to predict user preferences accurately within the given dataset [9].

5.1.3.1 Potentially skewed data

A P@K score of 0.6167 from the country model for customer X, and 0.4500 for the same model for customer Y indicates potential data bias, as this high score may imply that certain user buying patterns are disproportionately represented. These users might consistently purchase the same two or three products, thereby skewing the overall dataset towards these patterns [36]. In an attempt to further diagnose the issue, data plots 5.1 and 5.2 were generated based on customer X's data, representing the distribution of unique users per country, as well as the total number of orders

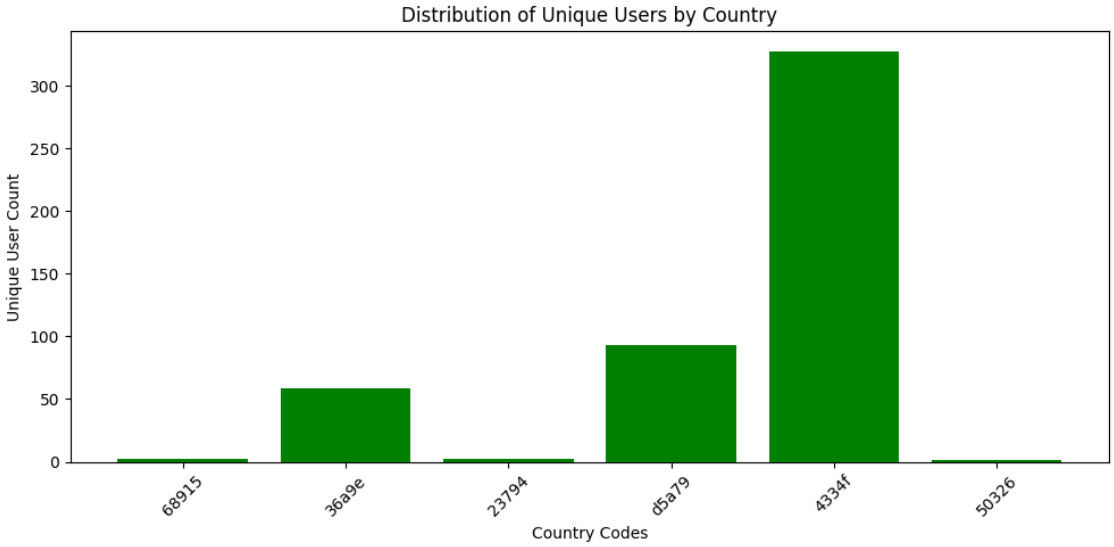

Figure 5.1: The bar chart of the distribution of users by country in customer X's website.

per country. However, no further research was made. They are included here for potential reference of future research.
To enhance the system further, it is imperative to address potential biases prior to model training. Strategies such as eliminating over-represented users or products from the dataset or employing techniques like down-sampling and over-sampling should be considered. Such interventions are aimed at delivering more equitable recommendations across a broader user base, thereby mitigating the influence of over-represented groups [37].

5.1.4 Analysis constrained by ML.NET

The process of training a matrix factorization model using the `MatrixFactorizationTrainer.Fit` method as provided by ML.NET is hindered by limited evaluation capabilities during training epochs. Specifically, the console output includes four columns labeled `'tr_sqerror+'`, `'tr_sqerror-'`, and `'tr_sqerror'`, which display numerical values purportedly representing different measures of training squared error. However, the lack of official documentation leaves these metrics ambiguous and their significance unclear [29].

This critical information is also inaccessible programmatically from within the application code, which hampers comprehensive analysis and understanding of the training process. Consequently, basing further analysis on these obscure console outputs is unreliable. Ideally, it would be beneficial to evaluate the model at each epoch, enabling direct observation of phenomena such as overfitting with respect to our custom average P@K score metric. This approach would eliminate the need to predefine a fixed number of training epochs, allowing for the extraction of the best-performing model as soon as overfitting is detected during training [39].

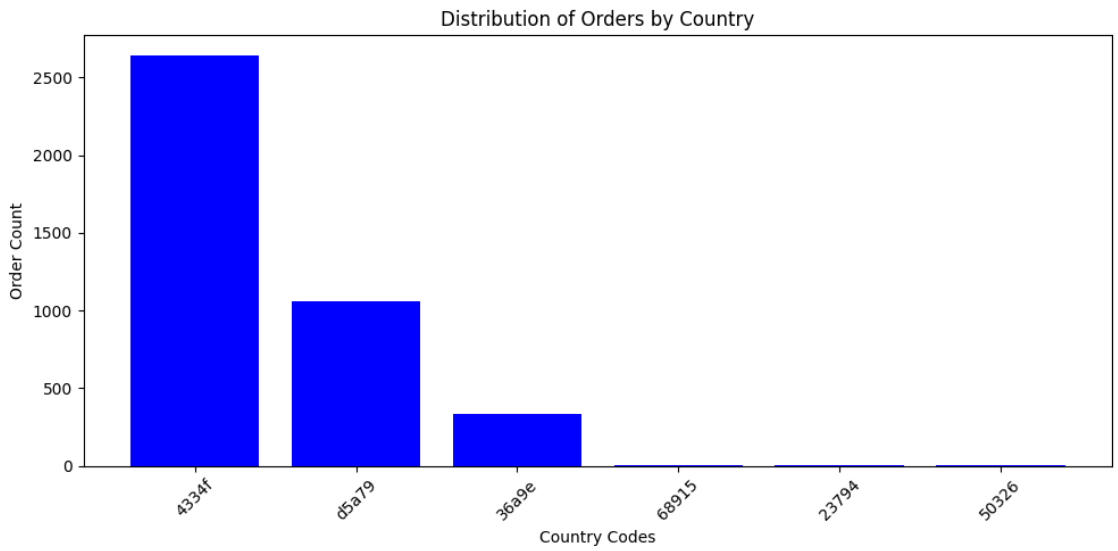

Figure 5.2: The bar chart of the distribution of orders by country in customer X's website.

Contrastingly, such functionality is readily available in Python libraries for matrix factorization, such as in the widely-used Surprise library, which allows for detailed monitoring and evaluation of model performance at each epoch [40]. This facilitates a more dynamic and responsive training process, enhancing the ability to optimize model performance effectively [41].

5.1.5 Finding the right value ranges for hyperparameters

To enhance the effectiveness of these parameters in future model iterations, a more rigorous analytical approach is recommended, as compared to the trial-and-error based approach outlined in this project. Understanding the precise impact of altering these parameters can lead to more targeted and efficient model tuning, potentially improving the model's performance across diverse datasets [38].

5.2 System architecture implications

5.2.1 The utilization of dependency injection within .NET

Adhering to the SOLID principles, every essential class was registered within the DI container through interfaces. This approach follows the dependency inversion principle, ensuring high cohesion and low coupling. Each class created within the application and registered within the DI-container was designed with a single purpose. Database access for instance, was managed through classes that are solely designed for that purpose. This modular design principle paired with the dependency injection created through the container is fundamental in providing a maintainable and scalable architecture [17].

5.2.2 Complexity management in system operations

The system's evolution from transient class-based implementation for each polling sequence, to a managed singleton class approach reflects significant improvements in efficiency and scalability. Centralizing control over polling instances reduces complexity and resource overhead. However, this design choice also centralizes failure points; issues in the manager class could disrupt polling across all users, impacting the overall system reliability [42].

The asynchronous operations for training, enhance the system's ability to manage concurrent requests effectively, improving user experience by reducing wait times. Yet, this concurrency introduces challenges in maintaining data consistency and handling error states effectively, which could compromise system integrity under high load conditions [43].

5.2.3 Security and integrity in data handling

Utilizing structured data models and MongoDB's schemaless databases helps maintain the flexibility of data management. These technologies facilitate rapid adaptation to changing data requirements without extensive schema redesigns. However, the schemaless nature of MongoDB can also lead to inconsistencies in data format, these inconsistencies can propagate errors through the system, affecting the accuracy of recommendations or the integrity of the system [44].

6
Conclusion

This project focused on developing a machine learning-based recommendation system and evaluate different approaches tailored for B2B e-commerce, specifically integrating it into the Parttrap ONE platform. The primary motivation was to enhance user experience and sales efficiency for Roima Intelligence's industrial clients through personalized product recommendations.

6.1 Key insights

6.1.1 Machine learning approach

The market basket analysis approach was considered but ultimately not selected due to its lack of personalization and its focus on global product associations rather than individual user preferences. It does not determine which product associations are most relevant for individual users, leading to repetitive and less relevant suggestions [46] as compared to collaborative filtering which is the preferred method for B2B e-commerce.

6.1.2 Precision at K (P@K) as the preferred metric

The initial reliance on Mean Absolute Error (MAE) proved inadequate for capturing meaningful interactions in sparse datasets, which are common in B2B e-commerce stores. Switching to Precision at K (P@K) provided a more accurate reflection of model performance by focusing on the relevancy of top recommendations. The P@K scores highlighted the model's potential and its limitations. For example, customer Y's UserID model achieved a P@K score of 0.1579, indicating that about 16% of recommended products were purchased.

6.1.3 Constraints of ML.NET

Using ML.NET for matrix factorization revealed limitations in evaluation capabilities during training, with ambiguous model training task output metrics hindering comprehensive analysis. The lack of detailed monitoring tools available in more established libraries like Python's Surprise impacted the ability to optimize model performance dynamically.

6.1.4 Dependency Injection and Scalability Management

Implementing dependency injection according to SOLID principles enhanced maintainability and scalability through high cohesion and low coupling. Transitioning to a managed singleton class for polling improved efficiency while introducing potential centralized failure points. Asynchronous operations for training improved the system's ability to handle concurrent requests, though high load data consistency requires further testing.

6.2 Future directions

Towards the future development, several key areas have been identified for enhancement. The following points outline the primary areas of focus:

- **Advanced metrics and evaluation**: Incorporating further evaluation techniques to better capture user-product interactions and improve recommendation precision.
- **Bias mitigation**: Implementing strategies like sampling to address data biases, ensuring equitable recommendations across diverse user bases.
- **Scalability improvements**: Exploring scalable solutions to distribute polling and training tasks effectively.
- **Comprehensive security measures**: Enhancing data security protocols to protect all forms of data, ensuring compliance with GDPR and CCPA regulations.

6.3 Final thoughts

This project successfully demonstrated the integration of a machine learning-based recommendation system within the .NET framework, highlighting both its capabilities and limitations. The findings emphasize the importance of appropriate evaluation metrics, bias mitigation, and advanced monitoring tools in optimizing model performance. The proposed future directions aim to build on these insights, driving further advancements in personalized recommendation systems for B2B e-commerce platforms.

Bibliography

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in *Proc. 20th Int. Conf. Very Large Data Bases (VLDB)*, Santiago de Chile, Chile, 1994, pp. 487-499.

[2] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, no. 2, pp. 281-305, Feb. 2012.

[3] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets," in *Proc. IEEE Int. Conf. Data Mining*, Pisa, Italy, 2008, pp. 263-272.

[4] R. Pan et al., "One-class collaborative filtering," in *Proc. Eighth IEEE Int. Conf. Data Mining*, Pisa, Italy, 2008, pp. 502-511.

[5] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, May-Jun. 2007.

[6] Microsoft, "ML.NET Documentation," 2021. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/machine-learning/. [Accessed: May 14, 2024].

[7] McKinney, W. (2010). Data Structures for Statistical Computing in Python. Proceedings of the 9th Python in Science Conference. Unpublished.

[8] S. Raschka, "Mlxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack," *J. Open Source Softw.*, vol. 3, no. 24, p. 638, 2018.

[9] G. Shani and A. Gunawardana, "Evaluating Recommendation Systems," in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds., Springer, 2011, pp. 257-297.

[10] X. Su and T. M. Khoshgoftaar, "A Survey of Collaborative Filtering Techniques," *Advances in Artificial Intelligence*, vol. 2009, Article ID 421425, 2009.

[11] M. Waskom et al., "Mwaskom/seaborn: v0.11.0 (September 2020)," Zenodo, 2020. [Online]. Available: https://zenodo.org/record/4043796. [Accessed: May 14, 2024].

[12] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[13] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate Research*, vol. 30, no. 1, pp. 79–82, 2005.

[14] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[15] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California, Irvine, 2000.

[16] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.
[17] R. C. Martin, *SOLID: Object-Oriented Design*. Clean Coders LLC, 2003.
[18] National Institute of Standards and Technology, "FIPS PUB 197: The Advanced Encryption Standard (AES)," 2001.
[19] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag, 2002.
[20] M. Masse, "REST API Design Rulebook," O'Reilly Media, 2011. [Online]. Available: https://www.oreilly.com/library/view/rest-api-design/9781449317904/. [Accessed: May 14, 2024].
[21] P. G. Campos, F. Díez, and I. Cantador, "Towards a more realistic evaluation: testing the ability to predict future tastes of matrix factorization-based recommender systems," *Semantic Scholar*, 2022. [Online]. Available: https://www.semanticscholar.org/paper/Towards-a-more-realistic-evaluation
[22] Microsoft, "ASP.NET Core documentation," Available: `https://docs.microsoft.com/en-us/aspnet/core/`. [Accessed: May 14, 2024].
[23] Microsoft, "ASP.NET Core GitHub Repository," Available: `https://github.com/dotnet/aspnetcore`. [Accessed: May 14, 2024].
[24] Banker, K. (2011). *MongoDB in Action*. Manning Publications.
[25] Chodorow, K., & Dirolf, M. (2010). *MongoDB: The Definitive Guide*. O'Reilly Media.
[26] Microsoft, "Dependency Injection in ASP.NET Core," *Microsoft Docs*, `https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection`.
[27] M. Seemann, *Dependency Injection in .NET*, Second Edition, Manning Publications, 2019.
[28] C. C. Johnson, "Logistic matrix factorization for implicit feedback data," in *Advances in Neural Information Processing Systems*, vol. 27, pp. 78-85, 2014.
[29] Microsoft, "ML.NET: Machine Learning framework," [Online]. Available: https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet. [Accessed: May 14, 2024].
[30] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30-37, Aug. 2009.
[31] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proc. 10th Int. Conf. World Wide Web*, Hong Kong, 2001, pp. 285-295.
[32] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.
[33] J. Davidson et al., "The YouTube video recommendation system," in *Proc. Fourth ACM Conf. Recommender Systems*, Barcelona, Spain, 2010, pp. 293-296.
[34] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 5-53, Jan. 2004.
[35] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, *Recommender Systems: An Introduction*. Cambridge University Press, 2010.

[36] T. Kamishima, S. Akaho, H. Asoh, and J. Sakuma, "Enhancement of the neutrality in recommendation," in *Proc. 7th ACM Conf. Recommender Systems*, Foster City, CA, USA, 2014, pp. 8-14.

[37] Y. Ge, X. Chen, H. Cui, and Z. Hu, "Bias and Debias in Recommender System: A Survey and Future Directions," *arXiv preprint arXiv:2010.03240*, 2020.

[38] S. Rendle, "Learning Recommender Systems with Adaptive Regularization," in *Proc. Fifth ACM Int. Conf. Web Search and Data Mining*, Seattle, WA, USA, 2012, pp. 133-142.

[39] Y. Bengio, "Practical Recommendations for Gradient-Based Training of Deep Architectures," in *Neural Networks: Tricks of the Trade*, 2nd ed., G. Montavon, G. B. Orr, and K. R. Müller, Eds. Springer, 2012, pp. 437-478.

[40] N. Hug, "Surprise: A Python Library for Recommender Systems," *Journal of Open Source Software*, vol. 2, no. 12, p. 217, 2017.

[41] S. Funk, "Netflix Update: Try This at Home," *SVD*, 2006. [Online]. Available: https://sifter.org/ simon/journal/20061211.html. [Accessed: May 14, 2024].

[42] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, Inc., 2015.

[43] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, Inc., 2017.

[44] P. J. Sadalage and M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 2012.

[45] Y. Chen, A. Ghose, and P. G. Ipeirotis, "The Impact of Artificial Intelligence on Consumer Purchasing Decisions," in *Journal of the Association for Consumer Research*, vol. 16, no. 3, pp. 34-58, 2012.

[46] Z. Huang, H. Chen, and D. Zeng, "Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering," *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 116-142, 2008.

API Documentation Appendix

Below is the documentation and parameter list of the API for the application developed in the project.

Polling

StartPolling

- **HTTP Method**: POST
- **Route**: /Polling/StartPolling
- **Description**: Initiates polling for a specified user with an optional polling interval.
- **Parameters**:
  - **userId** (string): The unique identifier of the user.
  - **pollingInterval** (int?): Optional. The interval in days between each polling request.

StopPolling

- **HTTP Method**: POST
- **Route**: /Polling/StopPolling
- **Description**: Stops polling for a specified user.
- **Parameters**:
  - **userId** (string): The unique identifier of the user.

SetPollingInterval

- **HTTP Method**: POST
- **Route**: /Polling/SetPollingInterval
- **Description**: Updates the polling interval for a specified user.
- **Parameters**:
  - **userId** (string): The unique identifier of the user.
  - **days** (int): The number of days for the polling interval.

Training

ForceTraining

- **HTTP Method**: POST
- **Route**: /Training/ForceTraining

- **Description**: Initiates a forced training for a specified user by utilizing the most recent polled data.
- **Parameters**:
  - **user** (string): The unique identifier of the user.

GetStatus

- **HTTP Method**: GET
- **Route**: /Training/GetStatus
- **Description**: Retrieves the training status for a specified token.
- **Parameters**:
  - **token** (string): The token identifying the training process.

User

AddUser

- **HTTP Method**: POST
- **Route**: /User/AddUser
- **Description**: Adds a new user to the application.
- **Parameters**:
  - **request** (Object: UserConfig): Illustrated in 3.11 .

RemoveUser

- **HTTP Method**: POST
- **Route**: /User/RemoveUser
- **Description**: Removes a user and stops associated polling.
- **Parameters**:
  - **userId** (string): The unique identifier of the user to be removed.

UpdateUserPrefs

- **HTTP Method**: PATCH
- **Route**: /User/UpdateUserPrefs
- **Description**: Updates the preferences of an existing user.
- **Parameters**:
  - **request** (Object: UserConfig): Illustrated in 3.11.

Predictions

GetPredictionCustomer

- **HTTP Method**: GET

- **Route**: /Predictions/GetPredictionCustomer
- **Description**: Retrieves predictions for a customer based on the specified user ID and entity ID.
- **Parameters**:
  - **userId** (string): The unique identifier of the user.
  - **entityId** (string): The entity ID.
  - **top** (int): The number of top predictions to retrieve.

GetPredictionUser

- **HTTP Method**: GET
- **Route**: /Predictions/GetPredictionUser
- **Description**: Retrieves predictions for a user based on the specified user ID and entity ID.
- **Parameters**:
  - **userId** (string): The unique identifier of the user.
  - **entityId** (string): The entity ID.
  - **top** (int): The number of top predictions to retrieve.

GetPredictionCountry

- **HTTP Method**: GET
- **Route**: /Predictions/GetPredictionCountry
- **Description**: Retrieves predictions for a country based on the specified user ID and entity ID.
- **Parameters**:
  - **userId** (string): The unique identifier of the user.
  - **entityId** (string): The entity ID.
  - **top** (int): The number of top predictions to retrieve.

GetPredictionCity

- **HTTP Method**: GET
- **Route**: /Predictions/GetPredictionCity
- **Description**: Retrieves predictions for a city based on the specified user ID and entity ID.
- **Parameters**:
  - **userId** (string): The unique identifier of the user.
  - **entityId** (string): The entity ID.

  - **top** (int): The number of top predictions to retrieve.

GetPredictionZipcode

- **HTTP Method**: GET
- **Route**: /Predictions/GetPredictionZipcode
- **Description**: Retrieves predictions for a zip code based on the specified user ID and entity ID.
- **Parameters**:
  - **userId** (string): The unique identifier of the user.
  - **entityId** (string): The entity ID.
  - **top** (int): The number of top predictions to retrieve.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY