universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Synergistic fusion of session and sequence-based recommendation systems with dynamic graph neural networks“

verfasst von / submitted by

Simon Süwer

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 921
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von / Supervisor:	Ass.-Prof. Dr. Nils Kriege

Acknowledgements

I would like to take this opportunity to express my deep gratitude to all those who actively supported and constantly motivated me during the preparation of my Master's thesis.

My special thanks go to Ass.-Prof. Dr. Nils Kriege, who not only supervised my thesis, but also reviewed it. His valuable suggestions and constructive criticism were essential for the completion of this thesis.

I would also like to thank the University of Vienna. In particular, I would like to thank Lorenz Kummer, who stood by me with great patience, genuine interest, and untiring helpfulness. Our many stimulating discussions and exchanges of ideas have been instrumental in bringing this work to its present form.

I am deeply grateful to Refurbed, whose cooperation with the University of Vienna provided the basis for this master's thesis. Their willingness to share information and their insightful answers to my questions were invaluable.

Simon Süwer

Vienna, 21.08.2023

Abstract

In the age of online commerce, recommendation systems have become an important tool for businesses to provide personalized recommendations to users and increase sales.

Despite progress in hybrid recommendation systems, particularly in integrating sequential and session-based approaches, the use of Dynamic Graph Neural Networks (DGNNs) in this context still needs to be explored. This thesis addresses this research gap by evaluating the potential of DGNNs to improve the fusion of sequential and session-based recommendation systems. The objective is to generate more accurate personalized recommendations and provide a foundation for future scientific research in this area. This thesis combines session-based and sequential recommendation systems to achieve this objective.

The proposed Temporal Graph Recommendation (TGR) model uses DGNNs to capture and model user behavior in multiple sessions, improving the effectiveness of hybrid recommendation systems. TGR consists of three layers, which use graph-based methods and attention mechanisms to account for dynamic relationships between users, products, and interactions over time.

Crucially, the TGR model delivers relevant, personalized product suggestions derived from user-system interactions.

TGR outperforms state-of-the-art models and adds significant value to the evolving discourse on recommendation systems.

Kurzfassung

Im Zeitalter des Online-Handels sind Empfehlungssysteme zu einem wichtigen Instrument für Unternehmen geworden, um den Nutzern personalisierte Empfehlungen zu geben und so den Umsatz zu steigern. Obwohl hybride Empfehlungssysteme, die sequentielle und sitzungsbasierte Ansätze kombinieren, Fortschritte gemacht haben, ist der Einsatz von DGNNs in diesem Kontext noch weitgehend unerforscht. Diese Masterarbeit adressiert diese Lücke und evaluiert das Potential von DGNNs, um diese Ansätze effektiver zu kombinieren. Das entwickelte Temporal Graph Recommendation (TGR) Modell verwendet DGNNs, um das Benutzerverhalten über mehrere Sitzungen hinweg zu erfassen und zu modellieren, um die Effektivität kombinierter Empfehlungssysteme zu verbessern. Das TGR-Modell besteht aus drei Schichten, die graphbasierte Methoden und Aufmerksamkeitsmechanismen verwenden, um dynamische Beziehungen zwischen Benutzern, Produkten und deren Interaktionen über die Zeit zu berücksichtigen. Im Vergleich zu aktuellen Methoden zeigt TGR signifikante Verbesserungen und liefert relevante, personalisierte Empfehlungen. Diese Arbeit leistet einen wichtigen Beitrag zur Erforschung von Empfehlungssystemen.

Contents

List of Tables

List of Figures

List of Algorithms

Notations

Several mathematical variables are used in this thesis. They are defined in the following list. This listing contains only the most important notations, additional notations will be introduced further on. Additionally, some definitions may be explained in the corresponding paragraph for easier reading [Wu+20a; Wan+21b; Gao+21].

$\mathcal{I}$	Set of all items
$\mathcal{U}$	Set of all users
$\mathcal{E}$	Interaction between $\mathcal{U}$ and $\mathcal{I}$
$\mathcal{X}$	Input set where $(\mathcal{I}, \mathcal{U}, \mathcal{E}^{(u,i))} \in \mathcal{X}$
$\mathcal{M}$	Memory
$\mathcal{M}^{\mathcal{U}}$	Memory of seen users
$\mathcal{M}^{\mathcal{I}}$	Memory of seen items
$\mathcal{S}$	Set of all sessions
$\mathcal{S}^u$	Set of all sessions for user u
$\mathcal{S}_c^u$	Current session for user u $S_c^u \in \mathcal{S}^u$
$\mathcal{S}_h^u$	Historical session for user u $\mathcal{S}^u \setminus \mathcal{S}_c^u \in$
$\mathcal{G}^s$	Graph representation of $\mathcal{S}^u$
$\mathcal{T}$	Maximum timestamp within a sequence
t^c	Current timestamp of network $t^c \in \mathcal{T}$
$\mathcal{T}_{(u,i)}$	Timestamp of interaktion between u and i
$\mathcal{I}^p$	The set of items with position embedding

Notations

$\mathcal{A}$	The result of a self-attention network
$\mathcal{F}$	Self-observation
$s_g \backslash s_c$	global preference/current interest
h_v^l	Hidden state of node v embedded in propagation layer l
h_u^*	Final representation of user $u \in \mathcal{U}$
h_i^*	Final representation of item $i \in \mathcal{I}$
$\mathcal{N}_v$	Neighborhood set of node v
$\mathcal{W}$	Learnable transformation matrix
$\sigma(\cdot)$	Nonlinear activation function
$[\cdot;\cdot]$	Vector concatenation
$\hat{y}$	Score on items for user u
$\cdot_j, \cdot_k$	Indefinite graph indices
$\cdot^t$	Vector or matrix at time t
$\cdot_{id}$	ID for an feature (e.g. *user ID*: u_{id})

1. Introduction

With the proliferation of e-commerce and social media platforms, recommendation systems have become a preferred tool for many companies. A recommendation system can be defined as a filtering system designed to present personalized information to platform users. This promotes a positive user experience and business success [Wu+20a; Wan+21b]. A recommendation system can include a variety of functions, such as suggesting items in an e-commerce application or presenting multimedia content such as music, images, or videos on social networks or streaming platforms [Xie+21a; Wu+20a; Wan+21b]. In the modern world, more and more users rely on the recommendations provided by systems that reduce the number of available options from seemingly inexhaustible possibilities to a manageable subset. This can be considered an application of machine learning where the use case is related to a problem of significant importance in industry and academia. This requires the model to consider the user's historical interactions, including clicking, viewing, reading, and purchasing an item [Gao+21; Wan+21b].

The history of recommendation systems can be divided into three distinct periods [Gao+21]. The first phase was characterized by flat models, which included recommendation systems based on neighborhood methods. As mentioned earlier, these methods used the user's historical interactions to suggest objects similar to the user's past preferences [Che+16; Wu+20a; Gao+21; Wan+21b]. Neighborhood approaches are characterized by simplicity, efficiency, and effectiveness. These characteristics have led to the widespread use of this approach in practice [Wu+20a]. A complementary approach would be to design models that map users and objects in a common space using continuous vectors to make them directly comparable [Wu+20a]. As this approach has already made progress, the next step was the development of Neural Network models [Wu+20a]. Models based on NN have the advantage of exploiting nonlinear and nontrivial relationships between users and objects and can extend data sources to contextual, textual, and visual information. [Che+16; Wu+20a; Gao+21; Wan+21b]. NN methods have inherent limitations in capturing complex structural patterns present in the data during training [Wu+20a; Gao+21]. Challenges include the high dimensionality and sparsity of data,

the difficulty of managing long-term sequential dependencies, and the need for more explicit structural knowledge [Das+21]. These models are prone to overfitting when dealing with high-dimensionality and sparsity data [VF05]. NN also struggle to account for long-term relationships in sequential data and often do not encapsulate the inherent structural knowledge or rules that may exist in the data [BSF94]. As a result, the learned model may only consider the current user's historical interactions and may not capture the broader context. The last evolutionary step discussed in current research is using Graph Neural Networks (GNNs). These models address the abovementioned problem by using embedding propagation to iteratively aggregate neighborhood embeddings. In this approach, each node can access the information of its neighbors regardless of its position in the graph, capturing not only first-order but also higher-order information. [Che+16; Wu+20a; Gao+21; Wan+21b]. A first-order node is limited to receiving information only from its immediate neighbors [Sch+22]. However, a higher-order node can collect information from more distant nodes. This ability increases the potential of GNN to recognize more complex patterns and relationships within the graph [Sch+22]. Recommender systems use the graph structure to represent the relationships and connections between users, objects, and their interactions. Data are represented as a graph, with nodes representing users and objects and edges representing relationships or interactions. This structural representation allows recommendation systems to use the information embedded in connections and interactions to generate relevant recommendations. The advantage of this data structure in the context of recommendation systems is that it allows information to be extracted from the relationships between nodes within the graph. In addition, the graph structure enables the integration of external information sources, such as social relationships between users. In the case of information related to articles, this extends beyond the articles themselves to include the relationships between individual data points [Wu+20a; Gao+21; Wan+21b]. The primary benefit of this data structure is the ability of recommendation systems to analyze relationships between users and items to uncover patterns, similarities, and preferences [Wu+20a; Gao+21; Wan+21b]. Using techniques such as graph mining, graph embedding, and graph-based algorithms, recommender systems gain the ability to capture dynamic patterns of user behavior, enabling the derivation of relevant and personalized recommendations [Wu+20a; Gao+21; Wan+21b]. The graph structure is a powerful modeling tool to handle the large and diverse data found in the recommendation engines [Wu+20a; Gao+21; Wan+21b]. However, it is important to note that a graph alone cannot generate recommendations, as it is only an abstract representation of the relationships between users, objects, and

their interactions [Wu+20a; Gao+21; Wan+21b]. It is through careful use of the graph structure that the recommender systems' algorithms and models can generate meaningful recommendations [Wu+20a; Gao+21; Wan+21b].

1.1. Research Challenges

The combination of session-based and sequential recommendation systems has received considerable attention in recent years and has been studied by several authors [Lv+19; Mit+22; KML18; Zho+18; QKH17; Hid+18]. This combination enables the generation of personalized recommendations based on user behavior and context, leading to higher user satisfaction and motivation. In addition, it can improve the accuracy of recommendations and enable the generation of infrequent and non-obvious recommendations. Given the increasing interest in personalized recommendations and the availability of large amounts of data sources, it is crucial to explore the combination of session-based and sequential recommendation systems to develop new insights and recommendation algorithms [KML18; Zho+18; QKH17; Hid+18]. Session-based recommendation systems analyze user behavior within a single session to generate personalized recommendations, while sequential recommendation systems consider the entire history of user behavior to make predictions [Wu+20a; Gao+21; Wan+21b].

Research has shown that hybrid recommendation systems, such as the combination of session-based and sequential recommendation systems [SS21], can provide promising results, making it a viable approach to creating personalized recommendations [Bur02; E Ç17; Phu19; Guo20; Zha21]. Despite the progress that has been made, the study of combining session-based and sequential recommendation systems remains an open research area with numerous challenges [SS21; Bur02; E Ç17]. One fundamental problem is the lack of data, as large and high-quality datasets are needed that cover both session-based and sequential user interactions [Son+21]. Another issue is data representation, as merging disparate data sources and understanding user behavior require effective methods [Gao+21; Wu+20a]. Developing appropriate representations that adequately account for the dynamic nature of user sessions and action sequences is critical [Fen+19]. Complex information must be captured and adequately represented in machine learning formats [Gao+21; Wu+20a]. In addition, model complexity is a challenge, as the integration of sequential and session-based recommendation systems requires highly complex models and these models must adequately account for the many aspects of user interaction and require the use of advanced machine learning and data processing techniques [Bur02; E

Ç17; Phu19; Guo20; Zha21]. Evaluating and benchmarking combined session-based and sequential recommendation models is also challenging. Developing appropriate metrics and benchmarks is critical to objectively compare different approaches and evaluate their effectiveness [Bur02; E Ç17; Phu19; Guo20; Zha21]. In general, these challenges highlight the need for further research and innovation to realize the full potential of combining session-based and sequential recommendation systems. Overcoming these challenges will improve the effectiveness of these approaches in real-world application scenarios and enable a new level of personalized recommendations [KML18; Zho+18; QKH17; Hid+18].

1.2. Contribution

To take advantage of both session-based and sequential approaches, DGNNs are utilized. These approaches take advantage of the flexible representational capabilities of graphs to capture complex relationships and interactions in the data. DGNNs can effectively model the evolution of user sessions and the sequence of actions [Liu+21; Wu+20b]. They capture the relationships between the elements of the session and the temporal relationships between actions. As a result, they can develop a more complete comprehension of user behavior and generate more accurate recommendations [Li+20; ZWL20]. An important advantage of DGNNs is their ability to efficiently leverage knowledge from previous sessions and interactions. By integrating information from previous sessions, they can provide contextual recommendations that better match individual preferences and the current user context [Liu+21; Wu+20b; Li+20; ZWL20]. The purpose of this thesis is to apply DGNNs to the combination of session-based and sequential recommendation systems. To achieve this goal, a model is developed, presented in chapter 3, to analyze and model user behavior within an arbitrary number of sessions. To provide a comprehensive foundation for this thesis, a review of the existing literature and the underlying theoretical concepts is conducted in chapter 2. Based on this review, the innovative model will be presented and tested through experiments. These experiments will evaluate the effectiveness of DGNNs in predicting user behavior within the given problem. The results of this research will contribute to a deeper understanding of the application of DGNNs to the combination of session-based and sequential recommendation systems and lay the foundation for future research in this area.

2. Background

This chapter introduces different GNN-based recommendation systems to provide a basis for understanding the following implementation in chapter 3. Section 2.1 starts with a general overview of how recommendation and GNN-based recommendation systems work. Since this thesis focuses on graph-based recommendation systems, this chapter introduces the basics of GNNs to understand how graph information can be used for the recommendation system use case presented in section 2.2. The advantages of these systems are discussed in subsection 2.2.6. This thesis specifically focuses on developing and analyzing recommendation systems based on time-dependent factors. These systems are discussed in detail in section 2.3 and section 2.4. Finally, section 2.5 explains how systems can be tested and evaluated, and section 2.6 summarizes the topics covered in the previous chapters.

2.1. Recommendation systems

It is necessary to specify the concept of recommendation systems and their characteristics. In principle, a recommendation system can be defined as a special variant of a filtering system designed to provide personalized information to the user. From the provider's perspective of these systems, the objective is to promote profitability for a specific interaction. Personalized recommendations allow companies to target their items or services more effectively to customers, thereby increasing sales. In addition, recommendation systems can help increase user time on the platform, making it more attractive to advertisers [Gao+21]. These systems can be formally described as: $y_{u,i} = f(h_u^*, h_i^*)$ [Gao+21]. $i \in \mathcal{I}$ stands for an arbitrary item, $u \in \mathcal{U}$ stands for a user of the system. The rating function $f(\cdot)$ returns the user's (u) preference score for the item i, usually returned as a probability. The function $f(\cdot)$ can be implemented in different ways, for example, by a Neural Network, which will be the case in this thesis [Gao+21]. This method takes h_u^* as a representation of users and h_i^* as a representation of items. These representations can also be result vectors of previous computational steps, which follow a multistage

architecture procedure [Gao+21].

The subsection 2.1.1 describes how a typical recommendation system workflow is structured. This is followed by subsection 2.1.2, a description of the different scenarios of the recommendation system, including the sequential, session, bundled and knowledge graph recommendation systems. The key concepts specific to each scenario are explained in each scenario. The subsection 2.1.3 outline the general goals of the recommendation systems, which are related to different levels of the recommendation process. Finally, subsection 2.1.4 covers various applications of these systems.

The general basics of GNNs are explained in section 2.2, and DGNNs is outlined in subsection 2.2.5. Additionally, the multi-head attention mechanism is explained in subsection 2.2.7. Different activation functions are defined in subsection 2.2.8, and a concise summary about Gated Recurrent Units (GRUs) is given in subsection 2.2.9.

2.1.1. Phases

A recommender system can be divided into three typical phases. The matching followed by ranking and final re-ranking. These three phases can be formed as a standard pipeline. Each of these phases differs in the input, output, and processing of the data itself. Depending on the use case, there may be additional or different phases, as this is only a possible representation [Gao+21].

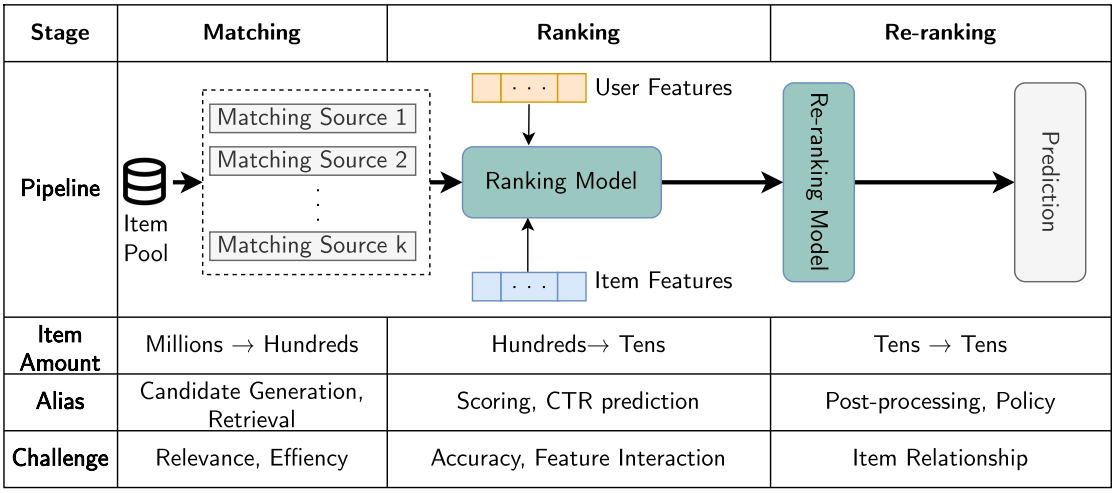


Figure 2.1.: A pipeline of recommendation systems, which can be viewed as a foundation for development.
Source: Own illustration, based on: [Gao+21]

According to Figure 2.1, the pipeline initially starts with a dataset containing potentially millions of items. The primary objective of the three stages is to generate targeted recommendations, thereby reducing the dataset to a small number of items [Gao+21].

The matching stage significantly reduces the data by using efficient models to approximately pre-select relevant items for the user. These preliminary models identify items that are likely to be of interest to the user, allowing subsequent models to operate more efficiently. Multiple models can be applied, including those based on geographical data or popularity [Gao+21].

In the following phase, the previously selected items from multiple models in the matching phase are combined and re-evaluated using a single model. The purpose of this model is to further reduce the number of potential items. Since the input set for this algorithm is much smaller than that used in the matching phase, it becomes possible to incorporate rich features such as user profiles and item attributes. It is critical to use only one model in this phase to ensure its suitability for the task and to include the necessary interaction between characteristics [Gao+21].

The final phase, re-ranking, aims to further optimize the small list of items obtained from the ranking phase. Its goal is to satisfy additional business requirements by changing the order of the item list or removing items that may result in redundant information. Alternatively, re-ranking can be optimized according to specific business requirements [Gao+21].

2.1.2. Scenarios

In research, recommender systems are classified into different scenarios depending on the intended application or the underlying data [Gao+21]. The following subsections present different scenarios for recommender systems. The presentation does not claim to be exhaustive but represents a selection of scenarios [Gao+21] which are relevant for this thesis and will be discussed in the further course of the thesis. Specifically, the following overview describes sequential recommendation, session-based recommendation, bundle recommendation, and knowledge graph-based recommendation scenarios [Gao+21]. In the following work, the focus will be on sequential recommendations and session-based recommendations. The bundle recommendation and the knowledge graph-based recommendation are presented as a contrast to clarify the emphasis of time-series-based systems (sequential, session-based) and to differentiate these scenarios more from the other scenarios [Gao+21].

Sequential recommendation

Recommender systems are frequently used in services that are characterized by a high number of user interactions over a long period of time [Gao+21]. The fundamental purpose of such systems is to use these past interactions to predict possible future interactions [Wu+20a]. These historical interactions can be conceptualized as time series that evolve over time. Using these time series, it is possible to predict the content that a user is likely to consume in the future [Cha+21]. As a result, the recommendation system can provide personalized recommendations to each user, taking into account their unique interests and preferences [Wu+20a; Gao+21; Cha+21]. A time series refers to a series of measurements of the same interaction object over time. In general, a time series is modeled as a sequence of random variables, and this stochastic process can be mathematically described as [Gam17]:

$$\mathcal{X}_{\mathcal{T}} = (x_0, x_1, ..., x_{\mathcal{T}}) = (x_t : t \in \mathbb{N}^{\leq \mathcal{T}}) \tag{2.1}$$

Where $\mathcal{T}$ denotes the length of the time series, while x_t denotes a particular measurement at a particular time in the time series. It should be noted that unlike general time series, the time series at hand are always finite. In the context of this thesis, the time series of user interactions is modeled as follows:

$$\mathcal{I}_{\mathcal{T}} = (i_1, i_2, ..., i_{\mathcal{T}}) = (i_t : t \in \mathbb{N}^{\leq \mathcal{T}}, i \in \mathcal{I}) \tag{2.2}$$

Note that Equation 2.2 assumes that there is only one user. In practice, however, there is usually at least one time series per user. In contrast to Equation 2.1, there are no random variables $\mathcal{X}_{\mathcal{T}}$, but interactions with items, which are denoted as $i \in \mathcal{I}$.

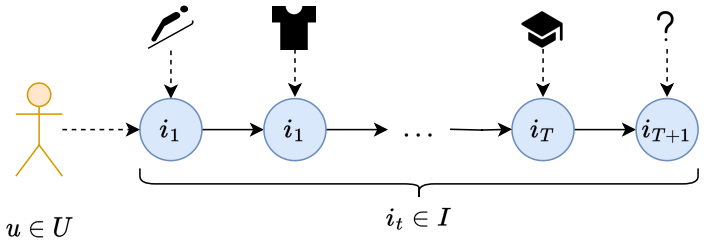


Figure 2.2.: An exemplary representation of a sequential recommendation. Based on historical interactions, for example in an online store, of a user, the recommendation system tries to predict the next item for the specific user.

The goal of sequential recommendation is to recommend the item that best matches the user's interests and preferences, which is defined as recommending the item $i_{\mathcal{T}+1}$. The mathematical clarification is illustrated in Figure 2.2. The consideration of time stamps is a fundamental aspect of sequential recommendation, since the sequence of a user's interactions with items plays an important role in predicting his future interests [Wu+20a; Gao+21; Cha+21]. Within-sequence modeling can be achieved through various approaches, such as Markov models, Recurrent Neural Networks (RNNs), or attention mechanisms. These approaches can help to capture the short- and long-term interests of the user and to account for possible dependencies between different elements of a sequence [Wu+20a; Gao+21; Cha+21].

However, information overload is a major challenge for sequential recommendation, as a user can quickly lose interest or overview if there are too many suggestions. One way to address this challenge is to limit the number of items suggested or to implement a diversity strategy to ensure better coverage of the user's different interests [Wu+20a; Gao+21; Cha+21].

In addition, there are two challenges that must be considered in the sequential recommendation. First, an item may appear in multiple sequences and a user may have multiple sessions, so relationships between sequences must be considered [Wu+20a; Gao+21; Cha+21]. A sequence can consist of one or more sessions. Second, it is important to consider modeling within a sequence to cover the user's short-term, long-term, and dynamic interests simultaneously. Some research demonstrates a positive effect when each session of a sequence is viewed separately, as this allows different interests to be viewed per session [Wu+20a; Gao+21; Cha+21].

Session-based recommendation

Session-based recommendations are often considered a subset of sequential recommendations. This is illustrated in particular by Figure 2.3, which also have temporal dependencies and are considered session-based assumptions [Wu+20a; Gao+21; Cha+21]. Session-based recommendation is useful in certain use cases where it is not possible to store all historical data due to storage limitations. Additionally, another use case for session-based recommendation is services where data is anonymous or users interact with the system too infrequently, such as retail [Wu+20a; Gao+21; Cha+21]. The data for this type of recommendation can, therefore, be seen as a short series of interactions by anonymous users. In this approach, the interactions of users in each session have

only session-based characteristics, and since there are no historical data, each session is considered independently [Wu+20a; Gao+21; Cha+21].

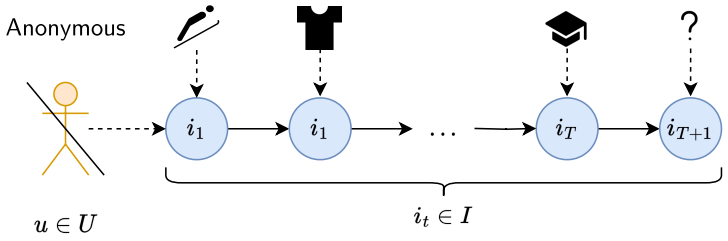


Figure 2.3.: An exemplary representation of a session recommendation. Based on historical interactions, for example in an online store, of a anonymous user, the recommendation system tries to predict the next item for the anonymous user.

Bundle recommendation and knowledge graph-based recommendation systems

The bundle recommendations focus on recommending a combination of items to the user as a whole. This is in contrast to traditional recommendation systems, which mainly focus on recommending individual items [Wu+20a; Gao+21]. There are three types of relationships in bundle recommendations. In the user-item relationship, each user interacts with multiple items. In the user-bundle relationship, users choose bundles. In the bundle-item relationship, a bundle consists of multiple items [Wu+20a; Gao+21]. Examples of bundle recommendations include adding songs to a playlist or composing articles [Wu+20a; Gao+21]. A Knowledge Graph (KG) is a structured model that expresses the relationships between elements using attributes. [Wu+20a; Gao+21]. In this context, a KG can be viewed as a data graph that accumulates the knowledge collected. The nodes in the KG represent entities of interest, while the edges represent the relationships between these entities [Hog+21]. The advantages of integrating Knowledge Graphs (KGs) into recommender systems include the fact that semantic relationships between elements of a KG allow deeper exploration of their connections, thus refining the representation of elements [Wu+20a; Gao+21]. Furthermore, elements with which a user has interacted in the past can be linked to the recommended elements, enhancing the interpretability of the results [Wu+20a; Gao+21]. These two variants will not be discussed further in this thesis, but should make it clear that there are a number of different variants, in addition to the two on which this thesis focuses [Wu+20a; Gao+21].

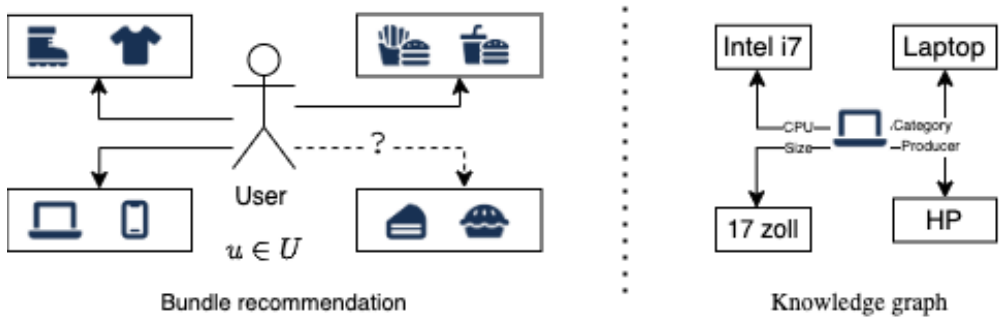


Figure 2.4.: Comparison of bundle recommendation and knowledge graph based recommendation systems. Left: exemplary illustration of bundle recommendation, Right: exemplary illustration of knowledge graph
Source: Own illustration, based on [Wu+20a; Gao+21]

Figure 2.4 illustrates the procedure of both systems.

2.1.3. Objectives

For a recommendation system, accuracy is central and there are other aspects that should characterize a recommendation system, such as diversity and explainability. In the following part, some of these aspects will be presented, although this list is not exhaustive [Gao+21].

Diversity

Diversity and balance are key concerns for recommendation systems. These concerns include diversity at the user level and at the system level. The objective at the user level is to avoid suggesting items that are too similar, as this could be counterproductive and not encouraging the user to explore the system [Gao+21]. If users are presented with only a narrow subset of items, they may not have the opportunity to explore the entire system. At the system level, the objective is to balance recommendations between popular items and long-tail items to provide users with an even coverage of all items [Zhe+21; Gao+21].

The figure in Figure 2.5 represents the diversity at the individual level as well as the diversity at the system level [Gao+21]. As illustrated, both forms of diversity exist. However, two challenges must be considered. The first challenge revolves around the

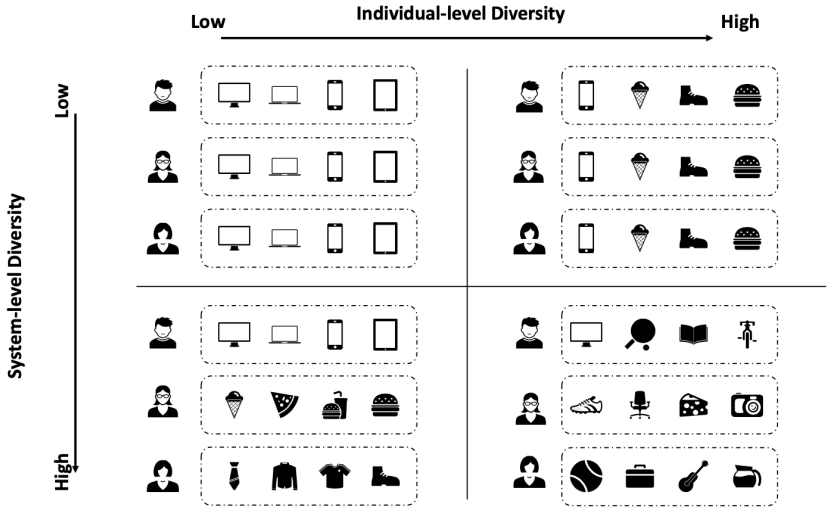


Figure 2.5.: Illustration of diversity at the individual level contrasted with diversity at the system level.
Source: [Gao+21]

significant variation in user interest, as indicated in Figure 2.5. Certain users may be more concerned with electronics, while others may prioritize food-related items. This discrepancy leads to significant variability in the strength of the signals of different items. The second aspect concerns the potential conflict between recommendation diversity and accuracy. Therefore, achieving a delicate balance between diversity at the individual and system level remains a key concern [Zhe+21; Gao+21].

Explainability

One of the primary objectives is to increase the explainability of recommendation systems [Gao+21]. There are two key aspects to this objective [ZC20; Gao+21]: providing users with an understanding of how and why specific articles are suggested to them, and enabling system operators to rapidly develop and improve these recommendation systems [ZC20; Gao+21]. Given the predominant machine learning orientation of such systems, achieving this objective is a significant challenge that has yet to be fully addressed. Current research pursues two different approaches: the first approach relies solely on explicable models, avoiding the use of *black box* techniques [ZC20; Gao+21], while the second approach aims at constructing models that can explain the results of *black box*

recommendation systems. This second approach is commonly referred to as explanation mining [ZC20; Gao+21].

2.1.4. Applications

Naturally, recommendation systems are also application-centric. For instance, a recommendation model designed for an e-commerce platform needs to focus on business value, emphasizing actions such as adding products to a shopping cart or making a purchase [Meh+18; Gao+21]. On the other hand, a streaming platform requires a model that aims to maximize user engagement by encouraging longer and more frequent use of the platform.

There are numerous other examples, such as recommendation systems used in social media platforms, job portals, or food marketing platforms. Each of these applications has unique goals that must be considered when developing the recommendation system [Meh+18; Gao+21].

2.2. Graph neural networks

This thesis focuses on graph-based recommendation systems highly based on GNNs. This section is intended to ensure a complete understanding of GNN by introducing various GNN variants such as Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), and Gated Graph Neural Networks (GGNNs). In the first part of this section, a comprehensive definition of graphs is given in subsection 2.2.1, which serves as a basic concept for understanding GNNs. Then, the core of subsection 2.2.2 is addressed by giving an overview of different existing GNN techniques. This overview reflects the current state-of-the-art for GNNs. A detailed definition of DGNNs is given in subsection 2.2.5, as it is a fundamental concept for understanding the model presented in this thesis. Section 2.2.7 introduces attention layers as an important technique in this context and further explains how attention layers enhance the GNN capability to identify intricate data patterns, thus improving recommendation generation. The justification for using GNNs in recommendation systems is highlighted in subsection 2.2.6, and explains how GNNs can recognize complex patterns in data and improve the quality of recommendations based on these patterns. The text emphasizes the advantages of GNNs, such as superior pattern recognition and adaptability, over traditional recommendation generation methods.

2.2.1. Graph structure

A graph is an abstract structure representing objects and their connections [You+20]. It is defined as follows $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ [You+20]. In this definition, $\mathcal{V}$ denotes the set of nodes, and $\mathcal{E}$ denotes the set of edges. A connection between nodes v_j and v_i is represented as $e_{i,j} = (v_i, v_j) \in \mathcal{E}$ [Wu+20a; Gao+21]. The neighborhood of a node v is denoted by $\mathcal{N}_\mathcal{V} = \{u \in \mathcal{V} | (v, u) \in \mathcal{E}\}$ [Wu+20a].

Graphs can be classified into several categories based on their properties. The following categories can be distinguished:

Directed Graph In a directed graph, or digraph, the edges have orientation. This can be expressed mathematically as $\mathcal{E} \subseteq (v, u) | (u, v) \in \mathcal{V}^2 \quad \text{and} \quad v \neq u$ [You+20; Wu+20a; Gao+21].

Undirected Graph An undirected graph is a special case of a directed graph where every edge is bidirectional [You+20]. This means an edge between two nodes can be traversed in either direction. An undirected graph is defined by the pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of undirected edges. Each undirected edge $(u, v) \in \mathcal{E}$ connects the nodes u and v without a specific direction [You+20; Wu+20a; Gao+21].

Homogeneous Graph A homogeneous graph is a graph in which each edge connects nodes, with only one type of nodes and edges [You+20]. The definition already presented in general applies $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. In a homogeneous graph, an edge can exist only between two different nodes [You+20; Wu+20a; Gao+21].

Heterogeneous Graph In a heterogeneous graph, each edge connects nodes, similar to a homogeneous graph. However, a heterogeneous graph can have different types of nodes and edges, representing different relationships between nodes. The definition $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ also applies here [You+20; Wu+20a; Gao+21].

Hypergraph A hypergraph is a generalization of a graph. In such a graph, each edge can connect any number of nodes [You+20; Wu+20a; Gao+21]. Formally, a hypergraph is defined as follows: $\mathcal{H} = (\mathcal{V}, \mathcal{E}_\mathcal{H})$. In this case, $\mathcal{V}$ is a set of nodes, and $\mathcal{E}$ is a set of hyperedges. Each hyperedge is a subset of $\mathcal{V}$, which means that $\mathcal{E}_\mathcal{H}$ is a subset of the power set of $\mathcal{V}$ ($\mathcal{E}_\mathcal{H} \subseteq \mathcal{P}(\mathcal{V})$) [You+20]. Hypergraphs can be much more complex than the graphs already introduced since an edge can connect any number of nodes.

As a result, hypergraphs can be useful in representing complicated relationships between different entities in a system [You+20; Wu+20a; Gao+21].

Dynamic/Static Graphs Dynamic and static graphs are categories that describe the state of a graph [Wu+22], DGNNs are explained in more detail in subsection 2.2.5. While a static graph remains unchanged, the properties and topology of a dynamic graph can change over time. These changes can affect the connections between nodes, for example, by adding or removing nodes or edges. Changes in attributes can affect the properties of nodes or edges, such as changing the size, weight, or color of nodes or edges. However, it is important to note that dynamic and static graphs are not mutually exclusive, as any graph can theoretically have both dynamic and static properties [You+20; Wu+20a; Gao+21].

In addition to the types of graphs mentioned above, it is notable that this list is not exhaustive. Other types of graphs, such as signed graphs, have been developed depending on the task at hand. These graph categories are generally orthogonal, meaning that a combination of types is possible. For example, a graph can be dynamic, directed, and heterogeneous [You+20; Wu+20a; Gao+21].

GNN Structures

In addition to the graphs' structures, discussed in the previous subsection 2.2.1, GNNs can also be categorized. Typically, these models are divided into two main structures: spectral and spatial models [You+20; Gao+21]. These are presented in the following list:

Spectral GNN Model In spectral models, the graphs are treated as signals and processed accordingly [Gao+21]. As the name implies, a graph convolution is performed in the spectral domain. This is done by first transforming the graph signals into the spectral domain using a Fourier transform and then applying a filter to the result. The results are then transformed back into the spatial domain. Formally, this can be described as $\mathcal{F} \star x = \mathcal{F}^{-1}(\mathcal{F}(g) \odot \mathcal{F}(x))$. Here $\mathcal{F}$ is the Fourier transform, f is a filter, and s is a signal [Gao+21].

Spatial GNN Model In spatial models, aggregation occurs directly at the level of graph structures [Gao+21]. Spatial models typically use a convolutional process that aggregates information from each node's neighbors to represent the node [Gao+21]. As such, they are somewhat similar to Convolutional Neural Networks (CNNs) in that they also allow weighted aggregation of local features [Gao+21].

Both models use the same principle of iterative collection of neighborhood information. This is crucial for detecting higher-order correlations between the nodes and edges of the graph. Dividing models into spectral and spatial types is useful for solving different problems more efficiently when applying GNNs. The choice of model depends on the specific task and the graph properties to which the model is applied [Gao+21].

2.2.2. GNN foundations

As mentioned in the previous section, the primary purpose of a GNN is to iteratively gather feature information from neighboring nodes in a graph and combine it with the target embedding, whether it is a node or an edge. This process is repeated layer by layer, resulting in updated graph embeddings [Wu+20a; Wan+21b; Gao+21].

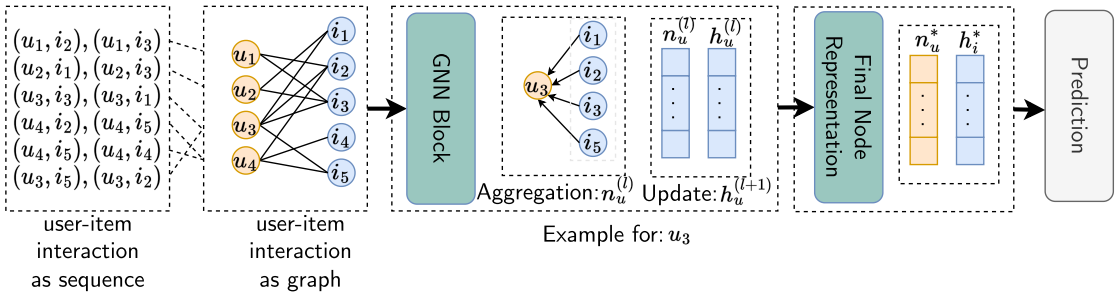


Figure 2.6.: The generalized representations of the process of GNN in collaborative filtering of user items in the context of recommendation systems.
Source: Own illustration, based on: [Wu+20a]

This section describes the basic structure and construction of GNNs and presents improved GNN models. A GNN consists of several propagation layers (l) stacked on top of each other [Wu+20a]. Each of these layers has two operations: the aggregation operation and the update operation. Generally, a layer can be represented by the equation Equation 2.3.

$$\begin{aligned} \text{Aggregation:} \quad & n_v^{(l)} = \mathit{Aggregator}_l(\{h_u^l, \forall u \in \mathcal{N}_v\}) \\ \text{Update:} \quad & h_v^{l+1} = \mathit{Updater}_l(h_v^{(l)} n_v^{(l)}) \end{aligned} \tag{2.3}$$

Here, h_u^l represents the node u in layer l. In the aggregation step ($\mathit{Aggregator}_l$), the neighbors are processed using a pooling operation (*mean*, *sum*, *min* or *max*) or an attention mechanism, depending on the model and the work [Wu+20a]. The neighborhood

of a node v is represented by $\mathcal{N}_v$. In contrast, the update step ($Updater_l$) integrates the representation of the central node with the result of the aggregation step (aggregated neighborhood) [Wu+20a].

Based on this basic structure, several improved models relevant to recommendation systems are presented. The selection in this list is derived from the referenced sources [Wu+20a; Gao+21] and should not be seen as a conclusive review of the literature.

GCN

The Graph Convolutional Network (GCN) is a well-established spectral model that combines graph convolution and Neural Network (NNs) [Zha+19b]. The model aims to perform semi-supervised graph classification by approximating the first-order Laplacian eigenvalue decomposition of the graph, which is used to aggregate information from its neighbors [Zha+19b; Wu+20a; Gao+21]. The Equation 2.3 presented earlier can be customized as follows:

$$\begin{aligned} \text{Aggregation:} \quad & n_v^{(l)} = \sum_{j \in \mathcal{N}_v} d_{vv}^{-\frac{1}{2}} \widetilde{a}_{vj} d_{jj}^{-\frac{1}{2}} h_j^{(l)} \\ \text{Update:} \quad & h_v^{l+1} = \delta(\mathcal{W}^{(l)} n_v^{(l)}) \end{aligned} \tag{2.4}$$

In Equation 2.4 and the following models, δ represents a nonlinear activation function unless otherwise specified. An activation function could be ReLU, defined in Equation 2.18, while other activation functions are defined in subsection 2.2.8. Furthermore, $\mathcal{W}^{(l)}$ corresponds to the learnable transformation matrix for layer l. In this context, $\widetilde{a}_{vj}$ is the adjacency weight and 0 unless node v is connected to j or $j = v$, and $djj = \sum_k \widetilde{a}_{kj}$ [Zha+19b; Wu+20a; Gao+21].

GraphSAGE

GraphSAGE is a spatial GNN model [HYL17]. In general, this model queries the neighbors of the target node and merges the embeddings of the target node with the summarized embeddings of the neighbor nodes to update the target node [HYL17]. The *Aggregator* uses a pooling operation (*mean*, *sum*, *min* or *max*) and a concatenation operation to update [Wu+20a; Gao+21].

$$\begin{aligned} \text{Aggregation:} \quad & n_v^{(l)} = Aggregator_l(\{\{h_u^l, \forall u \in \mathcal{N}_v\}\}) \\ \text{Update:} \quad & h_v^{l+1} = \delta(\mathcal{W}^{(l)} \cdot [h_v^{(l)}; n_v^{(l)}]) \end{aligned} \tag{2.5}$$

In the Equation 2.5, all the provisions that have been presented for Equation 2.3 apply [HYL17; Wu+20a; Gao+21].

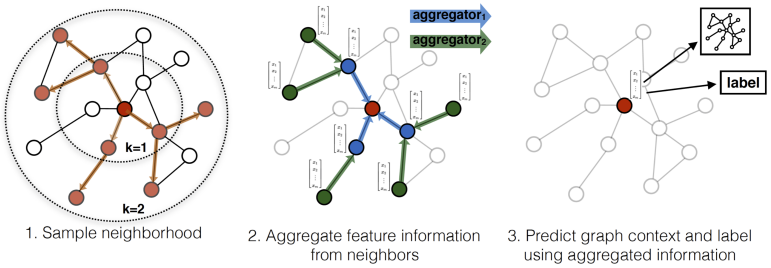


Figure 2.7.: Visual representation of the approach of *GraphSAGE*. Especially the sampling and aggregation approach.
Source: [HYL17]

As illustrated in Figure 2.7, *GraphSAGE* trains a separate embedding vector for each node, like the other models presented here. During this process, the feature information is aggregated from the local neighborhood of a node, with each *Aggregator* function obtaining this information from a different number of hops or search depths away from a given node [HYL17; Wu+20a; Gao+21].

GAT

Graph Attention Network (GAT) is a spatial GNN that addresses several challenges of spectral models, as described by Veličković et al. [Vel+17]. For example, it addresses the problem of poor generalization of a certain graph structure to other models [Vel+17; Wu+20a; Gao+21]. The GAT model assumes that the influence of neighbors on the central node is not identical and that the structure of the graph is not given [Vel+17]. Consequently, the contributions of neighbors are differentiated by exploiting the attention mechanism, and the vector of each node is updated taking into account its neighbors [Vel+17; Wu+20a; Gao+21].

$$\begin{aligned} \text{Aggregation:} \quad & n_v^{(l)} = \sum_{j \in \mathcal{N}_v} a_{vj} h_j^{(l)}, a_{vj} \frac{\exp(Att(h_v^{(l)}, n_v^{(l)}))}{\sum_{k \in \mathcal{N}_v} \exp(Att(h_v^{(l)}, n_v^{(l)}))} \\ \text{Update:} \quad & h_v^{l+1} = \delta(\mathcal{W}^{(l)}, n_v^{(l)}) \end{aligned} \tag{2.6}$$

In addition to the provisions defined in Equation 2.4, $Att(\cdot)$ is an attention function

and stands typically for

$$\text{leakyReLU}(a^T[(\mathcal{W}^{(l)}h_v^{(l)};(\mathcal{W}^{(l)}h_j^{(l)}])$$

and a is a learnable parameter [Vel+17; Wu+20a; Gao+21]. Furthermore, $\mathcal{W}^{(l)}$ is responsible for the transformation of the node representations in the propagation step l [Wu+20a]. The function leakyReLU is defined in Equation 2.19.

GGNN

A graph-to-sequence network, including Gated Graph Neural Network (GGNN), allows information represented as a graph to be used with a sequence generator to produce output that benefits from the graph structure [Li+15]. GGNNs uses a Gated Recurrent Unit (GRU) for the update step [Li+15; Wu+20a; Gao+21]. GRU is defined in detail in subsection 2.2.9.

$$\begin{aligned} \text{Aggregation:} \quad n_v^{(l)} &= \frac{1}{|\mathcal{N}_v|}\sum_{j\in\mathcal{N}_v} h_j^{(l)} \\ \text{Update:} \quad h_v^{l+1} &= \text{GRU}(h_v^{(l)}, n_v^{(l)}) \end{aligned} \tag{2.7}$$

One challenge with GGNN is that it executes the recurrent function multiple times on all nodes, which can cause problems when applied to large graphs due to poor scalability [Li+15; Wu+20a; Gao+21].

HETGNN

Heterogeneous Graph Neural Network (HETGNN) is a spatial GNN model optimized for heterogeneous graphs, as proposed by Zhang et al. [Zha+19a]. Since a heterogeneous graph contains different types of nodes and edges, neighbors are first divided into subsets according to their types. An *Aggregator* function applied per neighbor type uses a combination of RNNs and *MEAN* operations. Furthermore, neighborhood information is aggregated according to the attention mechanism [Gao+21; Zha+19a].

$$\begin{aligned} \text{Aggregation:} \quad n_v^{(l)} &= \sum_{f_i\in\mathcal{F}_v} a_{v_i} f_i \\ \text{Update:} \quad h_v^{l+1} &= \delta(\mathcal{W}^{(l)}, n_v^{(l)}) \end{aligned} \tag{2.8}$$

In this context, the set of embeddings is denoted as $\mathcal{F}_{\sqsubseteq} = f_1^t(v) \cup f_2(v), t \in \mathcal{O}_v$, where $f_1(v)$ corresponds to the embedding of the content of v, $f_1^t(v)$ is the aggregated embedding based on the type and $\mathcal{O}_v$ represents the set of node types [Zha+19a].

HGNN

Hypergraph Neural Network (HGNN) is a spectral model that implements a hypergraph neural network. It encodes higher-order data correlations in a hypergraph structure. The convolutional layer of the hypergraph is defined as [Wu+20a; Gao+21]:

$$\begin{aligned} \text{Aggregation:} \quad & \mathsf{N}^{(l)} = \mathsf{D}_v^{-\frac{1}{2}} \mathsf{E} \mathcal{W}^0 \widetilde{\mathsf{D}}_e^{-1} \mathsf{E}^{\mathcal{T}} \mathsf{D}_v^{-\frac{1}{2}} \mathsf{H}^{(l)} \\ \text{Update:} \quad & \mathsf{H}^{l+1} = \delta(\mathcal{W}^{(l)} \mathsf{N}^{(l)}) \end{aligned} \tag{2.9}$$

In this model, E is the adjacency matrix of the hypergraph. Furthermore, D_v refers to the number of hyperedges the node contains, and D_e represents the degree of the edge, that is, how many nodes the hyperedge contains [Wu+20a; Gao+21].

As introduced in the beginning of this chapter, the similarities and differences of the presendet models are illustrated in Figure 2.8.

2.2.3. Final node representation

In GNNs, the use of aggregation and update operations results in a gradual transformation of the node representations according to the depth of the network [Wu+20a]. In this process, the node vector of the last layer $\mathcal{L}$ is often considered as the final representation of a node ($h_v^{(\mathcal{L})}$) [Wu+20a; Liu+23]. However, the representations in the different layers of GNNs emphasize different aspects of the node information [Wu+20a; Liu+23]. In particular, the representations in the initial layers mainly represent the characteristics of the individual nodes [Wu+20a; Liu+23]. In contrast, the representations in the deeper layers consider more information from the node neighborhood, giving more weight to messages transmitted over different links [Wu+20a; Liu+23].

Recent research has developed strategies to integrate representations from different layers to maximize the use of individual and neighborhood information [Wu+20a; Liu+23]. This provides a more complete and potentially more informative representation of the nodes in the graph [Wu+20a; Liu+23]. The most common pooling methods are illustrated in Equation 2.10 [Wu+20a; Liu+23].

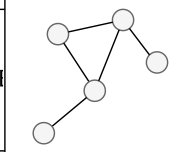
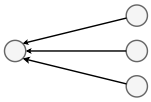
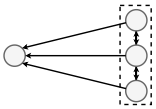
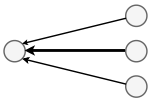
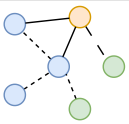
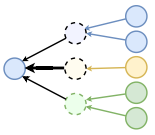
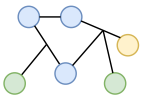
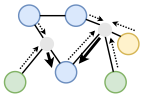
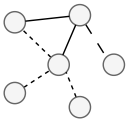
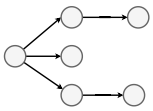
Model	Graph type of GNN	GNN category	Aggegrate	Equation
GCN		Spectral		Agg: $n_v^{(l)} = \sum_{j \in \mathcal{N}_v} d_{vv}^{-\frac{1}{2}} \widetilde{a}_{vj} d_{jj}^{-\frac{1}{2}} h_j^{(l)}$ Update: $h_v^{l+1} = \delta(\mathcal{W}^{(l)}, n_v^{(l)})$
GraphSAGE		Spatial		Agg: $n_v^{(l)} = \mathrm{Agg}_l(\{h_u^l, \forall u \in \mathcal{N}_v\})$ Update: $h_v^{l+1} = \delta(\mathcal{W}^{(l)}, n_v^{(l)})$
GAT		Spatial		Agg: $n_v^{(l)} = \sum_{j \in \mathcal{N}_v} a_{vj} h_j^{(l)}$ Update: $h_v^{l+1} = \delta(\mathcal{W}^{(l)}, n_v^{(l)})$
HetGNN		Spatial		Agg: $n_v^{(l)} = \sum_{f_i \in \mathcal{F}_v} a_{vi} f_i$ Update: $h_v^{l+1} = \delta(\mathcal{W}^{(l)}, n_v^{(l)})$
HGNN		Spectral		Agg: $\mathcal{N}^{(l)} = D_v^{-\frac{1}{2}} E W^0 \widetilde{D}_e^{-1} E^T D_v^{-\frac{1}{2}} H^{(l)}$ Update: $H^{l+1} = \delta(\mathcal{W}^{(l)} \mathcal{N}^{(l)})$
GGNN		Spectral		Agg: $n_v^{(l)} = \frac{1}{\lvert\mathcal{N}_v\rvert} \sum_{j \in \mathcal{N}_v} h_j^{(l)}$ Update: $h_v^{l+1} = \mathrm{GRU}(h_v^{(l)}, n_v^{(l)})$

Figure 2.8.: Comparison of several typical GNN models. For the graph type, the nodes and edges are represented by colors and lines. For aggregation, the line width indicates the weight of the neighborhood.
Inspired by: [Gao+21]

$$[!ht]\text{Mean-Pooling:} \quad h_v^* = \frac{1}{\mathcal{L}+1} \sum_{l=1}^{\mathcal{L}} h_v^{(l)} \tag{2.10}$$

$$\text{Sum-Pooling:} \quad h_v^* = \sum_{l=1}^{\mathcal{L}} h_v^{(l)} \tag{2.11}$$

$$\text{Min/Max-Pooling:} \quad h_v^* = \max_{\forall l \in \mathcal{L}} h_v^{(l)} \tag{2.12}$$

$$\text{Weighted-Pooling:} \quad h_v^* = \frac{1}{\mathcal{L}+1} \sum_{l=1}^{\mathcal{L}} \alpha^{(l)} h_v^{(l)} \tag{2.13}$$

$$\text{Concatenation:} \quad h_v^* = [h_v^{(0)}; h_v^{(1)}; ...; h_v^{(\mathcal{L})}] \tag{2.14}$$

In Equation 2.10, $\alpha^{(l)}$ is a learnable parameter [Wu+20a]. The list in Equation 2.10 is not exhaustive. It refers to the most common *Flat Pooling* methods [Liu+23]. Furthermore, there are other pooling methods, such as the *Node Clustering Pooling Methods* or the *Node Drop Pooling Methods*, which are not used in this thesis [Liu+23].

2.2.4. GNN optimization

This subsection explains how to optimize the models presented in subsection 2.2.2. GNN models can be divided into three types of tasks in which the models are used to solve graph-based classification, prediction, and regression problems [Gao+21]. Typically, these tasks involve up to three levels: Nodes, Edges, and Subgraphs [Gao+21]. These tasks include node classification in social networks, edge prediction in relational data, and graph classification for chemical compounds [Gao+21]. In regression tasks, continuous values are predicted instead of classification [Gao+21].

Since the presented models differ in the underlying tasks and their optimization, a generalized approach is presented in this section. In general, however, it is advisable to map and label relevant embeddings in order to formulate a comprehensible loss function [Gao+21]. Additionally, it is generally recommended to use existing optimizers for model learning [Gao+21]. As with most machine learning models, different mapping functions, such as the inner product, can be used, and a variety of loss functions are available for the model developer to choose from [Gao+21]. However, when using pairwise loss functions, it is important to distinguish between focusing on positive or negative samples [Gao+21]. Negative sampling is a method used to reduce computation time and memory requirements when training NNs [Arm+19]. Using negative examples can reduce the amount of training data without compromising the embeddings' quality [Arm+19]. Negative sampling is discussed in more detail in section 3.3.

To illustrate the model optimization process, the link prediction task is used as an example [Gao+21].

$$\begin{aligned} \mathcal{L} &= \sum_{p,n} -ln\sigma(s(p) - s(n)) \\ s(u,i) &= f(\{h_u^{(l)}\}, \{h_i^{(l)}\}) \end{aligned} \tag{2.15}$$

Formally, in Equation 2.15, $\sigma(\cdot)$ represents the sigmoid function defined in Equation 2.20. p and n refer to positive and negative samples, respectively, as described above. The function s is responsible for measuring the samples. The function s uses $f(\cdot)$ as a mapping

function [Gao+21]. Assume that the data $\mathcal{X} = (u, i, j)$ consist of positive and randomly selected negative samples observed [Gao+21]. For example, (u, i) or (u, j). In the context recommendation system, this would indicate that user u has interacted with item i but not with item j, where j is a negative sample of all other items that user u has not yet interacted with [Gao+21].

2.2.5. Dynamic GNN

Dynamic Graph Neural Network (DGNN) is a NN that operates on graphs and is capable of processing changes in the graph over time [Wu+22]. In dynamic graphs, the graph structure changes over time, unlike traditional static graphs, where the graph structure remains fixed [Wu+22; Kaz+19]. In this thesis, the DGNN architecture is the foundation for calculating the recommendations. Therefore, the explanation and principles of these models are crucial to understanding this thesis.

In general, DGNNs consists of layers of nodes, each representing the state of the graph at a given time. Forward propagation in DGNN calculates the state of the nodes in each layer as a function of the states of the neighboring nodes in the previous layer [Wu+22; Kaz+19]. This is described by Equation 2.16. To illustrate, consider a DGNN with three layers of nodes. In the first layer, each node represents the initial state of the graph [Par+19]. In the second layer, the state of each node is calculated as a function of the states of its neighboring nodes in the first layer. Similarly, in the third layer, the state of each node is calculated based on the states of its neighbors in the second layer [Par+19].

This forward propagation process allows the DGNN to capture both the spatial structure of the graph and the temporal changes in the graph over time[Par+19]. The result is a powerful model that can effectively process dynamic graph data, making it particularly useful for applications such as predictive recommendation systems [Par+19; Wu+22].

$$h_t^{(l+1)} = \sigma\left(\sum i \in \mathcal{N}_t \frac{1}{\sqrt{d_i d_j}} \mathcal{W}^{(l)} h_{i,t-1}^{(l)} + b^{(l)}\right) \tag{2.16}$$

In Equation 2.16, $h_t^{(l)}$ represents the state of the node in layer l at time t [Wu+22; Kaz+19]. $h_t^{(l)}$ is the state of the node in layer l at time t. The activation function σ is defined in Equation 2.20. The notation $\sum i \in \mathcal{N}_t$ describes the summation over the set of neighbors $\mathcal{N}_t$ for the node at time t [Wu+22; Kaz+19]. $\sqrt{d_i d_j}$ is a normalization factor, where d_i and d_j are the degrees of the nodes $v_i \in \mathcal{N}_t$ and $v_j \in \mathcal{N}_t$. This normalization

stabilizes the scaling of the feature values, which can help to speed up the convergence of the network [Wu+22; Kaz+19]. The expression $\frac{1}{\sqrt{d_i d_j}}\mathcal{W}^{(l)}h_{i,t-1}^{(l-1)}$ represents the weighted sum of the states of the neighboring nodes of the previous time step t, each multiplied by the weight matrix $\mathcal{W}^{(l)}$. This weight matrix is learned during the network training phase and determines the relationship between the nodes [Wu+22; Kaz+19]. $b^{(l)}$ is the bias vector for the layer l. It is learned with the weight matrix during training and allows the model to adjust the output shift [Wu+22; Zhu+22].

In general, DGNN is proving to be an extremely powerful graph data processing tool that has proven effective in various applications. For example, its ability to capture both global and local patterns in the data allows it to be used for predictive recommendation systems [Kaz+19]. In addition, DGNNs can handle data in various formats, including text, images, and audio [Zhu+22]. In particular, although DGNNs are highly flexible and suitable for a wide range of use cases, their effectiveness strongly depends on the data quality and the choice of hyperparameters. Consequently, careful modeling and optimization are essential to achieve optimal performance [Wu+22; Kaz+19].

Categories of dynamic graphs

Different prediction problems arise from problem definition, which different types of dynamic graphs can solve. For this reason, it is necessary to define these types [Wu+22; Kaz+19].

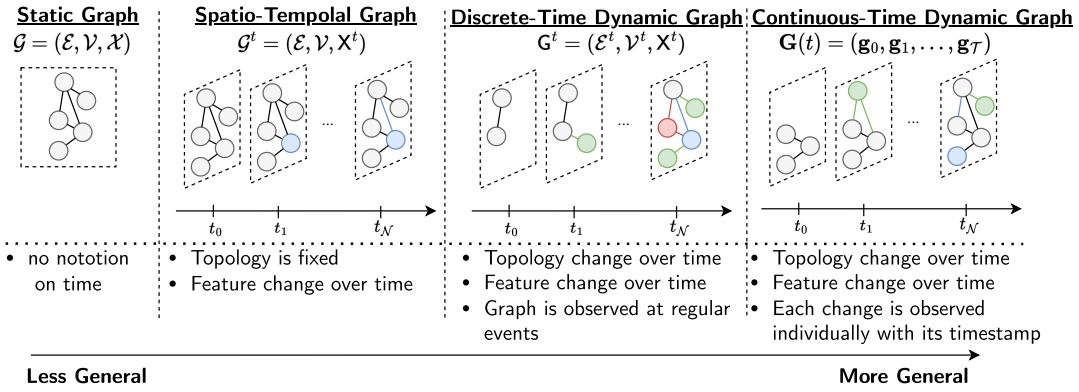


Figure 2.9.: Schematic representation of different categories of DGNNs, as opposed to static graphs
Source: Own illustration, based on: [Ros21]

In graph theory and its applications in data science and machine learning, the categor-

ization of graphs is of primary relevance [Wu+22]. Four categories commonly referred to in the literature include static graphs, spatio-temporal graphs, Discrete-time dynamic graphs (DTDGs), and Continuous-time dynamic graphs (CTDGs), as contrasted with Figure 2.9 [Wu+22; Kaz+19].

A static graph, defined as an ordered pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ is a set of edges, represents fixed relationships that persist consistently over time. In Figure 2.6, $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathsf{X})$ is used to compare the attributes of X more effectively with those of other types of graphs[Wu+22]. Static graphs are used in scenarios where temporal dynamics is absent or irrelevant to the analysis at hand [Wu+22; Kaz+19].

Spatio-temporal graphs build on static graph models by incorporating a temporal dimension [Wu+22]. The structure of the graph remains constant as $\mathcal{G}^t = (\mathcal{V}, \mathcal{E}, \mathsf{X}^t)$, but the attributes can change at a given time t. This type of graph is particularly useful for observing and analyzing temporal variations in systems where the structure remains unchanged [Wu+22; Kaz+19]. This is shown in blue in Figure 2.6.

In contrast, DGNN introduces an explicit dynamic component by representing it as a sequence of graph snapshots $\mathsf{G}^t = (\mathcal{V}^t, \mathcal{E}^t, \mathsf{X}^t)$ [Wu+22]. This approach models changes in both structure and attributes of the graph that occur in discrete time steps, a property essential in certain contexts such as social networks or collaborative filtering systems [Wu+22; Kaz+19]. In Figure 2.6, this is symbolized by green (indicating new nodes and edges) and red (indicating deleted nodes and edges).

Finally, CTDG is defined as the pair $\mathbf{G}(t) = (\mathbf{g}_0, \mathbf{g}_1, ..., \mathbf{g}_\mathcal{T})$ and describes the evolution of a graph in a continuous time frame [Wu+22; Kaz+19]. Unlike discrete dynamic graphs, where changes occur at fixed time intervals, this model allows the graph to change continuously at any time t. Here $\mathbf{G}(t)$ represents the graph at time t [Wu+22; Kaz+19]. The sequence $(\mathbf{g}_0, \mathbf{g}_1, ..., \mathbf{g}_\mathcal{T})$ represents the states of the graph over the entire time period considered, with each state $\mathbf{g}_t$ characterizing the graph at a particular time [Wu+22; Kaz+19]. The exact dynamics of the graph are described here by the function $\mathbf{G}(t)$ [Wu+22; Kaz+19].

In summary, the four graph categories are distinguished by their ability to model spatial and temporal relationships differently. The selection of the appropriate graph type depends heavily on the nature of the data and the specific requirements of the analysis [Wu+22; Kaz+19].

In the context of DGNN, there are two main approaches to modeling temporal dependencies: the *GNN-RNN* model and the *RNN-GNN* model. In the *GNN-RNN* model, an GNN is used first to model spatial dependencies in a graph at a given time [Wu+22;

Kaz+19]. The resulting embeddings allow RNN to model temporal dependency over multiple timesteps. Therefore, GNNs first extracts features from the graph, and RNNs uses these features to learn the temporal dependencies [Wu+22; Kaz+19]. This approach is particularly suitable when the structure of the graph at each time step is important and the temporal dependencies between these structures need to be modeled [Wu+22; Kaz+19].

The *GNN-RNN* model is in contrast to the *RNN-GNN* model, where a RNN is first used to model the temporal dependencies of the node features over multiple time steps [Wu+22; Kaz+19]. The resulting embeddings are passed to a GNN to model the spatial dependency in a graph at a particular moment [Wu+22; Kaz+19]. Therefore, RNNs first extracts temporal features from the nodes and GNNs uses these features to learn the spatial dependency in the graph [Wu+22]. When the temporal dependencies of the node features are more important than the graph's structure, this approach is particularly useful [Wu+22; Kaz+19].

2.2.6. GNN advantages for recommendations

During the past decade, the transition from traditional recommendation systems to models based on NNs has accelerated. The increasing use of GNNs in recommendation systems is substantial and is the focus of this discussion [Wu+20a; Gao+21]. The advantages provided by GNNs can be systematically divided into three primary domains, namely data structure, higher-order connectivity, and monitoring signal [Wu+20a; Gao+21]. The efficiency of GNNs is mainly due to the graph structure on which they are based. Most of the available data can be effectively represented as graphs, such as the connections between users with their profiles and items with their respective attributes collected by online platforms. In this scenario, users and items can be represented as nodes, with the interaction between the corresponding user and the item represented as edges [Wu+20a; Gao+21]. Additionally, a sequence of objects can be represented as a sequence graph, where each object is connected to one or more subsequent objects. Unlike recommendation systems that do not take advantage of the data structure, GNNs excel in learning representations, giving them a performance advantage [Cha+21]. This allows the creation of high-quality embeddings for users, items, and other features, thus improving the performance of the recommendation. However, the accuracy of the recommendation depends on the volume and precision of the data and its structure [Wu+20a; Gao+21; Wan+21b].

Traditional systems generally consider first-order connectivity only, primarily due to their reliance on training data consisting of interaction datasets. This limits the adaptive embedding space that describes the similarity between users and objects, which is critical for a high prediction quality [Wu+20a]. On the contrary, the GNN models have the ability to capture higher-order information through collaborative filtering [Wu+20a; Gao+21]. Collaborative filtering can be represented as a graph structure in terms of multi-hop neighbors. These neighbors are then incorporated into the learned representations through embedding and aggregation. Furthermore, the GNN models can incorporate signals into the learning process to improve performance. An effective example of such a signal is the target behavior, e.g. purchasing in the context of an e-commerce platform [Wu+20a; Gao+21]. To achieve this purpose, the system must utilize not only interactions that represent the target behavior, but also non-target behaviors such as item searching. These semi-supervised signals can be encoded in graphs, significantly improving recommendation performance [Gao+21; Wan+21b].

Another strength of GNNs is their ability to learn powerful representations of user and item information, a critical challenge for recommender systems [Wu+20a; Gao+21]. GNNs have demonstrated their ability to perform this task by representing and learning interactions in a graph structure, resulting in performance improvements [Li+21a]. These models can also efficiently manage different data structures. They are capable of handling both dense and sparse parameters and can adapt their performance accordingly, making them a versatile option for various recommendation systems [Li+21a].

Another important aspect is the integration of knowledge from Knowledge Graph (KG). GNNs can be combined with KG to address the *cold-start* problem inherent in collaborative filtering systems [Lyu+20; TOS20]. By integrating KG GNNs can capture explicit long-range semantics between users and items, while allowing for different edges between items [Lyu+20; TOS20].

In addition, GNNs can generate user-specific embeddings of articles by identifying important relationships in the knowledge graph for a given user. This enables personalized recommendations based on the user's specific interests and preferences [Wan+19]. Some GNNs use the smoothness assumption of the label, which states that neighboring articles in the knowledge graph are likely to have similar user relevance labels/scores. This assumption facilitates regularization of edge weights and improves the performance of the recommender system [Wan+19].

The scalability of GNNs is also important. GNNs have proven to be scalable for web-scale recommender systems [Yin+18]. For example, the PinSage model, a GNN-

based model, was trained on Pinterest using a graph with 3 billion nodes and 18 billion edges [Yin+18]. This shows that GNNs can scale to very large datasets while providing high-quality recommendations [Yin+18].

Another advantage of GNNs are their efficient training strategies. They can benefit from effective training strategies, such as using increasingly difficult training examples, to improve the robustness and convergence of the model [Yin+18].

In summary, the use of GNN in recommender systems is increasingly popular due to their ability to effectively exploit graph structure, capture higher-order connectivity, and integrate monitoring signals [Gao+21]. Furthermore, GNNs enable the incorporation of additional information, such as social network data, into the recommendation process, which can subsequently improve the accuracy of the recommendation [Wu+20a; Gao+21]. However, it is important to note that the accuracy of the recommendations depends on the quality and structure of the data [Wu+20a; Gao+21]. For this reason, engineers developing recommender systems should ensure that they select an appropriate data structure and integrate the necessary data into the graphs to achieve the best possible results [Wu+20a; Gao+21; Wan+21b].

2.2.7. Multi-Head-Attention-Mechanism

The multi-head attention mechanism has an important role in many DGNNs because it can model complex relationships in graphs [Kaz+19; Zhu+22]. The mechanism, proposed by Vaswan et al. [Vas+17], allows attention to be focused on different aspects of the graph, aggregating the relevant nodes to predict features and relationships [Kaz+19; Zhu+22]. This can capture dependencies between nodes in dynamic graphs and improve prediction accuracy. As this mechanism also has a significant role in this thesis, it is defined below.

Consider the input matrices $\mathcal{Q}$, $\mathcal{K}$, and $\mathcal{V}$, where $\mathcal{Q}$ is the query matrix, $\mathcal{K}$ is the key matrix and $\mathcal{V}$ is the value matrix. It can be said that the query vector $q_i \in \mathcal{Q}$ is used to calculate the attention of an element with respect to all other elements in the input [Vas+17]. The key vector $k_i \in \mathcal{K}$ is used to encode the relationships between the current element and other elements. The value vector $v_i \in \mathcal{V}$ contains the information that is aggregated when calculating the attention weights. Where i is the index of the element [Vas+17]. The mechanism consists of several parallel attentional layers. Each layer consists of a linear transformation of the input matrices, followed by an attention function, and a linear transformation of the output matrix [Vas+17]. For each layer i,

the attention function can be defined as follows

$$\mathcal{A}_i = \mathit{Attention}(\mathcal{Q}, \mathcal{K}, \mathcal{V}) = \text{softmax}(\frac{\mathcal{Q}_i \mathcal{K}_i^T}{\sqrt{d_k}})\mathcal{V}_i$$

The final output matrix $\mathcal{Y}$ is obtained by concatenating the results of each layer and subjecting them to another linear transformation:

$$\mathcal{Y} = \mathit{Concat}(\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_h)\mathcal{W}^O$$

where h is the number of layers and $\mathcal{W}^O$ is the weight matrix of the linear transformation. It is also important to note that the number of layers, the key and value matrices, and the dimensions of the matrices must be chosen to optimize the performance of the model [Vas+17]. d_k is the dimension of the key used to control the scaling, this is necessary because the product $\mathcal{Q}_i \mathcal{K}_i^T$ can become so large that softmax gradients are too small, which has a negative effect on training [Vas+17].

In conclusion, the multi-head attention mechanism computes a weighted sum of the value vectors for each element of the input, where the weights are determined by the similarities between the query vector and the key vectors [Vas+17]. By assigning a query vector, a key vector and a value vector to each input, the mechanism can model complex relationships between the elements in the input and aggregate relevant information [Vas+17].

2.2.8. Activation functions

Activation functions are an essential element of NNs because they apply a nonlinear transformation to the output of hidden layer neurons [DSC22]. This transformation is necessary for the modeling of complex nonlinear functions that are critical to the prediction of target variables from input data. If no activation functions are applied, the hidden layer neurons would compute only linear combinations of the input data, resulting in limited modeling capability [DSC22]. For this purpose, all activation functions used in this thesis are presented in this section to provide a central overview of them.

ELU Function

The Exponential Linear Unit function (ELU) is defined in Equation 2.17.

$$\text{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(\exp(x) - 1) & \text{if } x < 0 \end{cases} \tag{2.17}$$

The ELU function uses the α hyperparameter, which is set to 1 by default. The advantage of this function is that its derivative is smoother than that of the ReLU function, which allows for a better training result [DSC22].

ReLU Function

.

The Rectified Linear Unit function (ReLU) is defined in Equation 2.18.

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.18}$$

The ReLU function is a widely used activation function [DSC22]. The popularity of this function is mainly due to its fast computation and good scalability when many neurons are used. When using the ReLU function, it should be noted that some neurons may become *dead*, which means that due to their structure and underlying weighting, they remain unactivated and cannot produce any output [DSC22]. This phenomenon results in the network not being able to continue learning and therefore not being able to optimize [DSC22].

LeakyReLU function

The Leaky Rectified Linear Unit function (leakyReLU) is defined in Equation 2.19.

$$\mathrm{leakyReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{2.19}$$

The function leakyReLU uses the hyperparameter α, which is set to 0.01 by default [DSC22]. This function is similar to the ReLU function, but it does not have the disadvantage of *dead* neurons that can occur when using the ReLU function [DSC22]. In particular, the function leakyReLU has a linear slope for values less than zero, allowing even negative inputs to have a derivative, which has a positive effect on the trainability of the neural network [DSC22].

Sigmoid function

The sigmoid function (σ) is defined in Equation 2.20.

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2.20}$$

The σ function is a common activation function that has the important advantage of being a smooth and continuous function that provides a differentiable output for most input values [DSC22]. The output of the σ function is always restricted to the interval $[0, 1]$, which is particularly useful for binary classification problems where a prediction must be made in terms of *Yes* or *No* [DSC22]. However, for large input values, there is a risk that the output of the σ function will be close to zero or one, which is called *saturation*. As a consequence, the gradients can explode or disappear, which can make the neural network training much more complicated [DSC22].

Tanh function

The hyperbolic tangent function (tanh) is defined in Equation 2.21.

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{2.21}$$

The function tanh is similar to the function σ, but produces an output that is restricted to the interval $[-1, 1]$ [DSC22]. In effect, the tanh function is a smooth and continuous function that produces a differentiable output for most input values. In contrast to the σ function, the tanh function can also handle negative input values, making it more suitable for increasing the complexity of NNs [DSC22]. However, the function tanh can also produce *saturation*, making it difficult to train the NN. This is a well-known phenomenon with activation functions and can lead to problems such as gradients exploding or disappearing [DSC22].

Softmax Function

The function softmax is defined in Equation 2.22.

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{\mathcal{K}} \exp(z_j)} \tag{2.22}$$

The function softmax is an activation function that is used to transform the output of neurons into a probability distribution [DSC22]. Specifically, the function takes a vector z with elements $\mathcal{K}$ as input and outputs a vector with probabilities $\mathcal{K}$, where each probability is in the interval $[0, 1]$ and the sum of all probabilities is equal to 1 [DSC22]. These functions are often used in the last layer of NNs to predict a probability

distribution for classes [DSC22]. A significant advantage of the softmax function is that it is robust to small changes in input values. This leads to more stable predictions and is especially useful in practice [DSC22].

2.2.9. GRU

The Gated Recurrent Unit (GRU) that is used in this thesis is briefly described in this section. GRU is a special type of Recurrent Neural Network (RNN), introduced by Chung et al. [Chu+14], which is capable of storing information about past states and incorporate it into future predictions. Unlike a simple RNN, this model includes additional gating mechanisms that allow it to decide which information to keep and which to forget [Chu+14]. GRUs can be divided into three main components, which are briefly introduced below [Chu+14].

The update gate computes a vector with values in the interval $[0, 1]$, which information should be taken over from the previous state.

$$z_t^j = \sigma(\mathcal{W}_z x_t + \mathcal{U}_z h_{t-1})^j$$

Where h_{t-1} describes the previous state of the model, x_t is the current input vector, and σ is the sigmoid function. [Chu+14]

The reset gate determines what information to discard from the previous state [Chu+14].

$$r_t^j = \sigma(\mathcal{W}_r x_t + \mathcal{W} h_{t-1})^j$$

The hidden state is updated by integrating the current input vector and the previous state with the gates [Chu+14].

$$h_t^j = (1 - z_t^j) * h_{t-1}^j + z_t^j * \tanh(\mathcal{W} x_t + \mathcal{U}(r_t \odot h_{t-1}))^j$$

The new hidden state h_t is calculated by a weighted combination of the previous hidden state h_{t-1} and an updated hidden state, which is calculated by the activation function tanh based on a linear transformation of $r_t * h_{t-1}$ and x_t. [Chu+14]

The weight matrices $\mathcal{W}_z$, $\mathcal{W}_r$, and $W\mathcal{W}_r$ are network training parameters and are optimized during the training process. [Chu+14]

2.3. Sequential recommendation

Sequential recommendation systems, as introduced in subsubsection 2.1.2, are designed to predict future user activities based on their past behavior in a chronological sequence

[Gao+21; Wu+20a]. Unlike traditional recommendation systems, which use a static view of user preferences, sequential systems take into account the dynamic nature of a user's preferences [Gao+21; Wu+20a]. To illustrate, a user may prefer traditional dishes during the Christmas season, but tend to prefer lighter summer dishes during the summer. Therefore, a proficient sequential recommender system must be able to adapt to such transient or short-term changes in user preferences by taking into account time series interactions [Gao+21; Wu+20a]. The concept of GNN has received recent attention in the literature on sequential recommendation systems [Cha+21; Wu+20a; QCJ18]. These concepts provide an efficient method to capture the interdependencies among user interactions by modeling these interactions in the form of a graph [Cha+21; Wu+20a; QCJ18]. One way to graphically represent user interactions is through adjacency lists, where each node represents an item and each edge represents the sequential dependency between items [Cha+21; Wu+20a; QCJ18].

Section 2.3.1 will discuss the challenges of constructing a graph for this purpose and the appropriate methods for modeling user interactions. Then, information diffusion and sequential preferences will be discussed and GNN will be identified as a powerful tool to model user interactions and preferences.

The *cold-start* problem arises when there is limited or no interaction data available to new users. It is a common problem in recommender systems [Hu+19; Gao+21]. This challenge arises because the system has minimal or no knowledge of the new user's preferences or behavior, which can lead to inaccurate or irrelevant recommendations [Hu+19; Gao+21]. In the field of recommender systems research, the *cold-start* problem is often addressed by incorporating knowledge from other domains or by using unobserved samples to extract negative signals [TOS20; Bi+20]. Another practical solution might be to create a hybrid model that incorporates both sequential and non-sequential user and system information [Hu+19; Gao+21]. This approach would allow the system to better capture more comprehensive patterns of user behavior, leading to accurate recommendations even in situations with minimal or no interaction [Hu+19; Gao+21].

As illustrated in Figure 2.10 and in previous discussions, sequential recommendation systems use a set of items $\mathcal{I}$, where $i \in \mathcal{I}$ denotes a particular item. The user interacts with the system, producing a sequence of interactions of items: $i_1, ..., i_{\mathcal{T}}$. As described in chapter 2, $\mathcal{T}$ represents the number of interactions and $i_{\mathcal{T}}$ is the last item with which the user interacts [Cha+21; QCJ18]. The primary objective of this system is to predict the next item, $i_{\mathcal{T}+1}$. In general, the user's preferences are inferred from the implicit feedback provided by the chronological sequence [Cha+21; QCJ18].

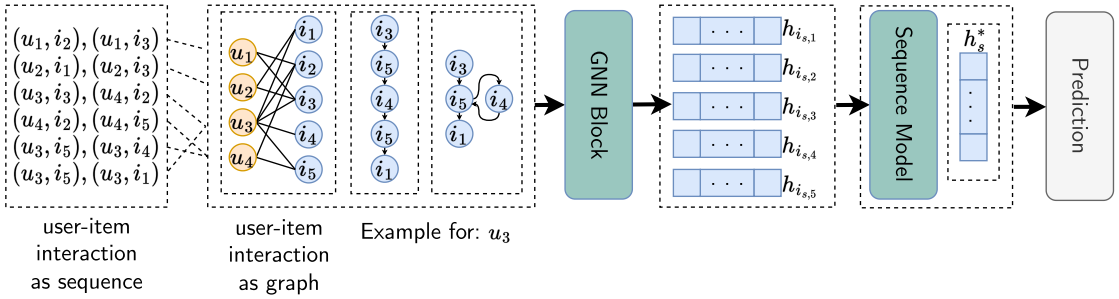


Figure 2.10.: The generalized representations of the process of GNNs in collaborative filtering in the context of recommendation systems. Adapted to the use case of sequential recommendation
Source: Own illustration, based on: [Wu+20a]

2.3.1. Graph construction

All the data discussed in this thesis follow the time series defined in subsubsection 2.1.2. These sequences are the foundation for the construction of a sequential graph, which is described in this section. Before discussing the specifics of sequence graph construction, the general principles of graph construction in the context of recommender systems are covered for a more complete understanding. No specific recommendations are proposed, only considerations for designing and structuring a graph.

Nodes Nodes are important in graph modeling and are equally important in GNN models. The definition of nodes significantly affects the scope of these models, with the majority of node parameters being occupied by layer 0 node embeddings. In the modeling process, a decision may be necessary as to whether different types of nodes, such as user nodes and item nodes, should be considered as distinct node types or as a common node type [Wu+20a]. In collaborative filtering, both approaches are feasible and should be addressed according to the use case and the structure of GNNs [Gao+21; Wu+20a]. Furthermore, numerical features, such as item prices, can be transformed into categorical features and represented as nodes. This allows the hierarchical structure of the information and the complex relationships between features to be captured, which is particularly important in applications such as recommendation systems, where capture the relationships between different features is key to the accuracy of the recommendation [Gao+21].

Edges Similar to node design, edge design is critical, as it greatly affects the quality of

the graph. The basic relationships between nodes can be represented as interactions between items [Gao+21; Wu+20a]. However, the edges in a graph can also represent relationships, such as the relationship between items and a category. The resulting graph density should be considered during graph construction. If a graph is too dense, methods such as sampling, filtering, or pruning the graphs are recommended to prevent the number of neighbors from becoming too large, which can lead to performance degradation and inferior results [Gao+21; Wu+20a]. On the contrary, a graph that is too sparse may not produce satisfactory results due to insufficient relationships [Gao+21].

While the previous paragraph proposed strategies for general consideration, this paragraph focuses specifically on the development of a sequence graph. The simplest approach is to convert each sequence directly into a directed graph, where each item is considered a vertex and an edge is established between them [Wu+20a; Gao+21]. For example, a sequence $i_1 \rightarrow i_2 \rightarrow i_3$ would be represented in a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = i_1, i_2, i_3$ and $\mathcal{E} = (i_1, i_2), (i_2, i_3)$. However, this approach runs the risk of creating a sequence with only a few items [Wu+20a; Gao+21]. A sequence graph constructed from a single short sequence has limited nodes and links. Consequently, models based on such sequence graphs may not contain enough knowledge to capture the dynamic preferences of users. The advantages of GNNs would not be applicable to this graph [Wu+20a; Gao+21].

A possible solution to this problem is to extend the structure of the sequence graph with additional data, which can be either from the same user or independent of the user. The selection in the following list outlines different structures that have been proposed in various papers, derived from reference sources [Wu+20a; Gao+21], and should not be considered as a conclusive review of the literature.

Each of these approaches has the independent advantage of incorporating more information into the graphs, thus improving the performance of the sequential recommendation system [Qiu+19; Qiu+20; Wu+20a].

Heterogeneous Graph Neural Network (HETGNN) As proposed by Zhang et al. [Zha+19a], this method constructs edges between two consecutive items in a given behavior sequence, using their respective behavior types as edge types [Zha+19a]. HETGNN uses all behavior sequences for its graph pre-construction strategy [Zha+19a; Wu+20a; Gao+21].

Physics-guided Neural Networks (PGNN) As detailed by Daw et al. [Daw+17], PGNN uses a user's known historical sequences of a user to complete the graph [Daw+17], facilitating the construction of a complete sequence graph [Daw+17; Wu+20a; Gao+21].

Global Context Enhanced Graph Neural Networks (GCE-GNN) GCE-GNN and Dual Attention Multi-Dimensional Integration (DAT-MDI) independently enrich the local context using historical user data as global context [Wan+20b]. In both methods, all data are given equal weight [Wan+20b; CGS21; Wu+20a; Gao+21].

Task Recommendation (TASRec) TASRec, which is proposed by Zhou et al. [Zho+21], uses historical data as a global context while focusing on more recent interactions [Zho+21]. The goal is to generate more accurate recommendations by weighing more recent interactions [Zho+21; Wu+20a; Gao+21].

Dual-channel Graph Transition Network (DGTN) This approach of Zheng et al. [Zhe+20], uses historical data as a global context and selects items from sessions similar to the current session for recommendation generation [Zhe+20]. DGTN aims to represent similar transition patterns more frequently [Zhe+20; Wu+20a; Gao+21].

Metapath Aggregated Graph Neural Network (MA-GNN) As proposed by Ma et al. [Ma+19], this method assumes that the last user interaction affects not only the next item, but also several items to come [Ma+19]. MA-GNN adapts the graph structure of the current sequence to include this last item and on this basis, adds, for example, three subsequent items to the graph along with their corresponding edges [Ma+19; Wu+20a; Gao+21].

Star Graph Neural Networks (SGNN-HN) This model introduces a virtual *star node* as the center of the sequence, as proposed by Pan et al. [Pan+20]. A *star node* is a virtual node that forms the center of the sequence and is therefore connected to every other node in the graph [Pan+20]. Each item of the graph can benefit from a general representation of the sequence through the vector representation of the *star node*. The model assumes that the relationships between distant items have negligible predictive value [Pan+20; Wu+20a; Gao+21].

Graph-less Neural Networks (LESSR) As proposed by Chen and Wong [CW20], this model, is a unique approach that constructs two graphs from a sequence [CW20].

One graph reflects the sequence of adjacent items, while the second provides a shortcut path that allows you to go from any item to any subsequent item. This design aims to mitigate the long-term memory inefficiency that can result from ignoring sequential neighbor information [CW20; Wu+20a].

Session Recommendation with Hypergraph Attention Networks (SHARE) This model takes advantage of the representational power of hypergraphs, as highlighted by Wang et al. [Wan+21a]. SHARE uses hypergraphs to represent higher-order relationships and integrate information. Each session has its own hypergraph, with hyperedges defined by sliding windows of different sizes. This approach provides rich information between sessions on item relationships [Wan+21a; Wu+20a; Gao+21].

Self-Supervised Hypergraph Convolutional Networks (DHCN) As highlighted by Xia et al. [Xia+20], DHCN uses the power of hypergraphs [Xia+20], such as SHARE [Gao+21]. However, instead of using multiple hypergraphs, *DHCN* adopts a single session-independent hypergraph structure. Session-to-session graphs are modeled based on hypergraphs, with each session represented as a node and edge weights computed based on shared items [Xia+20; Wu+20a].

Session-based Recommendation with Graph Neural Networks (SR-GNN) This recommendation system, as proposed by Wu et al. [Wu+19], models both users and items using a two-level graph [Wu+19]. SR-GNN uses historical and current interactions to complete the graph and generate a recommendation [Wu+19].

The above methods represent only a fraction of the possible approaches to graph building. As illustrated, in addition to the basic design of the graph, it is crucial to consider how the graph is constructed from a sequence [Wu+20a; Gao+21].

2.3.2. Information Gain

An essential aspect in the context of sequence graphs is the generation of information. User preferences often evolve, which requires a comprehensive comprehension of changing interests [Cha+21; Gao+21; Wu+20a]. For example, a user's primary interest may start out as electronics and then shift to books. Considering only the most recent user interactions risks providing an incomplete representation of user interests [Cha+21; Gao+21; Wu+20a]. Therefore, it is critical that the representation of interests encompasses the entire history of the user, not just a short period of time [Cha+21; Gao+21; Wu+20a].

Various recommendation methods have been developed to address this, with the pooling method widely recognized for its simplicity and intuitiveness. This technique is often used in the context of GNNs for sequential recommendations [Cha+21; Gao+21; Wu+20a]. For example, the GGNN method uses the pooling method to distribute information in directed graphs [Li+15].

The mean pooling method, which calculates the average value for each feature, is typically used to merge information from previous and subsequent nodes [Cha+21; Gao+21; Wu+20a]. A GRU can then integrate this information from neighbors and the central node. A major advantage of GRU over other RNNs, such as LSTM, is its ability to address the problem of vanishing gradient and provide superior performance [Cha+21; Gao+21; Wu+20a]. GRU is defined in detail in subsection 2.2.9.

The Equation 2.23 illustrates these considerations.

$$\begin{aligned} n_{i_{s,t}}^{in(l)} &= \frac{1}{|\mathcal{N}_{i_{s,t}}^{in}|} \sum_{j \in \mathcal{N}_{i_{s,t}}^{in}} h_j^{(l)} \\ n_{i_{s,t}}^{out(l)} &= \frac{1}{|\mathcal{N}_{i_{s,t}}^{out}|} \sum_{j \in \mathcal{N}_{i_{s,t}}^{out}} h_j^{(l)} \\ n_{i_{s,t}}^{(l)} &= [n_{i_{s,t}}^{in(l)}; n_{i_{s,t}}^{out(l)}] \\ h_{i_{s,t}}^{(l+1)} &= \mathrm{GRU}(h_{i_{s,t}}^{(l)}, n_{i_{s,t}}^{(l)}) \end{aligned} \tag{2.23}$$

In this context, $n_{i_{s,t}}^{in(l)} \in \mathcal{N}$ and $n_{i_{s,t}}^{out(l)} \in \mathcal{N}$ represent the neighborhood set of the preceding and following nodes, respectively [Cha+21; Gao+21; Wu+20a]. Furthermore, i is the index within the sequence of users, each index is assigned a time t.

$$n_{i_{s,t}k}^{(l+1)} = \mathrm{GRU}(h_{i_{s,t}}^{(l)}, n_{i_{s,t}}^{(l)})$$

This method improves previous approaches by preserving the order of items within the neighborhood, thus reducing information loss [Cha+21]. It differs from most methods that disregard item positions while using the permutation-invariant aggregation function during message passing [Cha+21]. However, some methods retain information about the position of items within their structures [Cha+21].

Another enhancement, proposed by Zhou et al. [Zho+18], is the use of an Attention Updating Gate (AUGRU) instead of GRU, which can be seamlessly integrated into the above procedures [Zho+18]. The scalability of AUGRU ensures that less relevant interests have minimal impact on the hidden state, thus promoting evenly developed relative interests and mitigating previous problems [Zho+18; Cha+21].

With the final embeddings, a prediction can be made using methods such as a two-layer neural network with feedforward functionality to estimate the next item the user will interact with [Cha+21; Xu+20b]. Equation (2.24) illustrates this.

$$\hat{y} = \text{Predict}(h_{i_{s,t},k}^{(l+1)}) \tag{2.24}$$

As mentioned previously, a major problem with analyzing sequences in a GNN is that long-range dependencies between nodes are not well captured [Cha+21; Wu+20a]. Thus, the representation of one item is not sufficient to successfully capture the next item. In addition, there is the loss of information that certain methods have when creating sequences in graphs [Cha+21; Wu+20a]. Various strategies have been developed to address this, in particular those that weight nodes within a sequence differently. This is usually done using an attention mechanism, as suggested in Equation 2.25 to Equation 2.26. The example of integration of the attention mechanism given here is merely illustrative. Other interesting techniques are suggested in the following text [Cha+21; Wu+20a].

The combination of the computed attentional weights between the last item and all other items in the sequence can be called the global preference. Integrating the global preference with the local preference yields the total preference [Wu+20a]. The problem here is that the total preference is highly dependent on the last interaction. Therefore, it is possible to stack multiple layers of self-attention to capture long-range dependencies, as is the case with *GC-SAN*. It is also possible to add a position embedding to preserve the relative order of items [Wu+20a; Cha+21].

$$\mathcal{P} = \{p_1, ..., p_{\mathcal{T}} | e_i = s_i + p_i\} \tag{2.25}$$

As mentioned above, the self-attention model does not know the positions of the items in the sequence $s_i \in \mathcal{I}$, therefore, as clarified in Equation 2.25, the positions are inserted [Xu+20b; Cha+21]. Here, an adaptive position embedding is used instead of a fixed position-dependent vector, $p_i \in \mathcal{P} \in \mathbb{R}^{\mathcal{T} \times d}$ [Xu+20b; Cha+21]. With this information, GRU can be used, for example, or AUGRU as suggested above. To illustrate the introduction, a mechanism of self-attention outside of GRU is illustrated in Equation 2.26, this is only for illustrative purposes, but would also tend to be possible [Xu+20b; Cha+21].

$$\begin{aligned} \mathcal{S} &= Attention(\mathcal{P}\mathcal{W}_q, \mathcal{P}\mathcal{W}_k, \mathcal{P}\mathcal{W}_v) \\ \mathcal{F} &= FFN(\mathcal{S}) = \mathrm{ReLU}(\mathcal{S}\mathcal{W}_1 + b_1)\mathcal{W}_2 + b_2 \end{aligned} \tag{2.26}$$

The *attention* refers to the attention layer in Equation 2.26, so the definition in subsection 2.2.7 applies. Furthermore, $\mathcal{W}_q, \mathcal{W}_k, \mathcal{W}_v \in \mathbb{R}^{d \times d}$ are the learnable parameters, where d is a hyperparameter. As the operation is linear, a feedforward network is used, typically using the GRU in the case of the models discussed. The parameters $\mathcal{W}$ and b, also introduced above, are learnable, where $b_1, b_2 \in \mathbb{R}^d$ [Xu+20b; Cha+21]. The ReLU is defined in Equation 2.18.

As demonstrated in the previous formula, the result in the self-observation example is $\mathcal{F} = f_1, ..., f_\mathcal{T}$. Each $f \in \mathcal{F}$ adaptively extracts information from the preceding items. In this process, $\mathcal{F}$ has two components, long-term preference and current interest [Xu+20b].

In this scenario, the long-term preference is defined as follows $s_l = \sum_{i=1}^{\mathcal{T}} \mathrm{softmax}(\mathcal{F}) f_i$. Current interest, denoted $s_c = f_\mathcal{T}$, represents the last item involved [Wu+20a; Gao+21]. The softmax, defined in Equation 2.22, is crucial in regulating the summation results [Xu+20b]. The two embeddings (s_l, s_c) can then be combined to form the final embedding of the sequence as $\mathcal{W}_c[(s_c; s_l]$, where $\mathcal{W}_c \in \mathbb{R}^{d \times 2d}$ is the linear transformation of the concatenation of s_c and s_l. This result can be combined with the information about the item to derive a score [Wu+20a; Gao+21]. However, as noted above, the presented process is merely illustrative. Most contemporary GNNs depend on GRU or AUGRU, as discussed above [Xu+20b].

2.3.3. Summary

In essence, sequential recommendation uses sequential historical user behavior to predict potential future interactions [Xu+20b; Cha+21]. Various papers propose different GNN-based methods to achieve this [Wu+20a; Gao+21]. Graph construction, which ranges from general recommendations to different ways of modeling a sequential graph, is an important topic covered in this section [Wu+20a; Gao+21]. The simplest construction adds an edge between two nodes. Approaches for handling sequences that are too short are also proposed [Wu+20a; Gao+21]. Several models and techniques are proposed to extract information from the sequence graph, the optimal method depending on the trade-off between computational cost and complexity of the sequence graph [Xu+20b; Cha+21]. The amount of data also influences the balance between computational cost and performance gain and the choice of model [Wu+20a; Gao+21].

The issue of sequential preference is then addressed. The author of this paper introduces an attention mechanism to incorporate item representations into the sequence [Cha+21]. This includes appending position embeddings that reflect the relative order of items to improve the final result [Cha+21]. The main models of sequential recommender systems, their sources and key features are summarized in Table 2.1.

2.4. Session recommendation

A common approach in recommendation systems is to use historical interaction data in a time series format to predict the next items in a recommendation [Hua+21; Qiu+19]. Unlike sequential recommendation systems, which assume, which assume that each user's sessions are known and continuously recorded. In many applications, it is not possible to identify the user, and therefore only a record of the user's behavior during an ongoing session is available [Gao+21; Wu+20a].

The primary purpose of session-based recommendations is to predict user interactions from anonymous sessions [Gao+21; Wu+20a]. The system models a session's limited amount of data to generate the most accurate recommendation possible [Hid+16]. In addition to user interests, session data may contain noisy signals [Gao+21; Wu+20a]. For example, consider the following sequence: $i_1^m \rightarrow i_1^m \rightarrow i_2^m \rightarrow i_3^m \rightarrow i_4^w \rightarrow i_5^m$. In this example, i_t^m represents an item in the *movie* category, while i_t^w represents an item in the *water* category [Gao+21]. Given this sequence, the interaction with the *water* item was likely unintentional, indicating a noisy signal, in addition to the primary interest of the *movies* [Gao+21]. Therefore, a major focus of graph learning is to distill the user's core interests, uninfluenced by noisy data [Gao+21].

To achieve these goals, a session graph is used to detect transitions that are difficult to identify by other means. Each session is typically used to build a global preference [Wu+20a; Gao+21; Wu+19]. This global preference can then be combined with the current interest of each session, using an attention network to obtain the most accurate results [Xu+19; Liu+18; Hua+21; Xu+19].

The objective of this section is to highlight the differences between these different approaches and sequential recommendations. In this case, the purpose is to differentiate recommendation strategies without repeating duplicate content. Therefore, the models used for the analysis will be kept from being repeated.

2.4.1. Session graph construction

The process of constructing session graphs is similar to the construction described in subsection 2.3.1. However, in this case, the purpose is to model a single session, so each session sequence s can be modeled as a directed graph $\mathcal{G}^s = (\mathcal{V}^s, \mathcal{E}^s)$ [Wu+19]. In this context, the suffix s denotes the current session. For the session graph, $v_{s,i} \in \mathcal{V}^s$ and for each edge $(v_{s,i-1}, v_{s,i}) \in \mathcal{E}^s$ [Wu+19]. In particular, the edge encapsulates the interaction between i_{t-1} and i_t [Wu+19]. Since interactions can occur multiple times, elements within a sequence can be repeated, and the weight of the edge can be adjusted accordingly to reflect that sequence [Xu+19; Liu+18; Hua+21].

Unlike sequential recommendations, session sequences are typically short, and user behavior tends to be limited. Consequently, a session graph derived from a single session will contain a limited number of nodes and edges [Wu+19; Qiu+19; Wan+20b; Gao+21].

To address this challenge, it is theoretically possible to apply any strategy proposed for a sequential recommendation, with the variation of using similar session data instead of data from the same user [Qiu+20; Zhe+20; Zha+21a]. This could be achieved by extending the relationships of the current session graph with related sessions or by creating a global graph to support the transition patterns in the current session using the global context [Gao+21; Qiu+20; Zhe+20; Zha+21a]. Furthermore, all the techniques from the previous section that do not require user-specific data are applicable [Gao+21]. Another technique proposed in the research is to assign a weight of 1 to a node if it does not loop back to itself [Xia+20; Wan+20b; CW21]. This idea is based on observations made in daily life and in the data set, suggesting that it is common for users to click on two consecutive links several times in a session [Gao+21].

A possible general procedure is shown in Figure 2.11.

2.4.2. Information Gain

Extracting information from session graphs involves a GNN. Fundamental methods like those discussed above are less suitable because session-based recommendation systems typically use weighted and directed session graphs [Gao+21]. Approaches like GCN and GAT are only appropriate for unweighted and undirected graphs [Gao+21]. As a result, these methods cannot be applied directly to session graphs [Gao+21]. Therefore, much of the current research uses an intentionally weighted graph layer, as proposed earlier [Gao+21].

The models are described in more detail below. A notable difference is the inclusion of

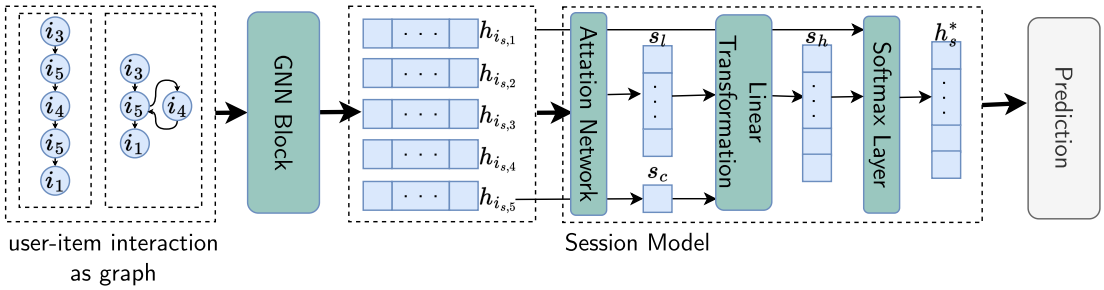


Figure 2.11.: The generalized representations of the process of GNNs in collaborative filtering of user items in the context of recommendation systems. Adapted to the use case of session recommendation
Source: Own illustration, based on: [Wu+20a; Wu+19]

a multilayer self-attention mechanism, which is particularly relevant in the context of a session graph, as suggested by recent research [Xu+19; Liu+18; Gao+21; Wan+20b]. Therefore, this concept is defined in Equation 2.27.

$$\begin{aligned} \mathcal{S}^1 &= \mathcal{S} = \mathit{Attention}(\mathcal{E}^s) \\ \mathcal{S}^k &= \mathit{Attention}(\mathcal{S}^{k-1}) \end{aligned} \tag{2.27}$$

Here, $\mathcal{S}^k \in \mathbb{R}^{n \times d}$ is the final output of the multilayer self-attention network. This output can be used as described in subsection 2.3.2. It is crucial to emphasize that this is merely an elaborate subject exposition. No model is presented or explained per se. It is a descriptive presentation designed to illustrate the underlying concepts and ideas without the use of a mathematically exact model [Xu+19; Liu+18; Gao+21; Wan+20b].

To propagate information about the constructed graph, one of the following four propagation mechanisms is most commonly used in session-based recommendations.

Gated GNN Gated GNN, as proposed by Li et al. [Li+15], uses modified gated recurrent units together with modern optimization techniques [Li+15]. The result is a flexible model similar to sequence-based models (e.g., GRUs), except that this model is tailored for graphs [Gao+21].

Graph Convolutional Networks Models based on GCNs, as proposed by Zhang et al. [Zha+19b], perform graph convolution on hypergraphs and line graphs [Zha+19b]. This has the advantage of representing a session from two different perspectives.

Depending on the model, graph convolution can also be performed on the dynamic graph or at the item level to capture the relationship between items [Zha+19b; Gao+21].

Graph Attention Networks Typically, models based on GATs, as proposed by Veličković et al. [Vel+17], extract information from the graph by stacking layers in which nodes can observe the features of their neighborhood [Vel+17]. Implicitly assigning different weights to different nodes in a neighborhood reduces the cost compared to models that require expensive matrix operations. In the context of session-based recommendation models, GAT is run on a directed session graph to capture contextual relationships of higher order between elements [Vel+17]. It is also possible to run GAT on local and global session graphs to capture all the information [Vel+17; Gao+21].

GraphSAGE GraphSAGE, proposed by Hamilton, Ying and Leskovec [HYL17], uses node features to efficiently embed nodes for previously unseen data [HYL17]. GraphSAGE is applied to the multi-relational item graph, uses this graph to capture information from neighbors, and supports different types of neighbors [HYL17; Gao+21].

2.4.3. Conclusion

In summary, the development of session-based recommendations is an ongoing research process. The research pursues different approaches to overcome the predicament of anonymous sessions [Gao+21]. One approach is to enrich the session graph with information and relationships between nodes or additional information and relationships from similar sessions, or all sessions conceptualized as a global context [Gao+21]. Other methods aim to add edges to the session graph to capture the intricate relationships and complexity of the information in the session data [Gao+21]. Other models and research are still trying to improve the recommendation by more thoroughly modeling higher-order relationships in the session data [HYL17; Gao+21]. Table 2.2 summarizes the most important models for session-based recommendation systems, listing their work sources and key characteristics.

2.5. Datasets

This section presents data sets that can be used to evaluate the performance of recommendation systems. These systems are widely used in various domains to provide

personalized recommendations for items, services, or content. To evaluate the accuracy of recommendation systems, it is essential to have appropriate data sets and evaluation metrics [Wu+20a; Gao+21]. These data sets are suitable for specific recommendation tasks due to their unique characteristics [Wu+20a; Gao+21]. Appropriate evaluation metrics for recommendations systems are also discussed.

2.5.1. Evaluation data sets

The following list contains frequently used data sets to evaluate sequential and session-based recommendation systems based on GNN [Wu+20a; Gao+21]. A detailed description of each data set used for the experiments in this thesis can be found in chapter 4. The following list provides a summary.

YooChoose Originally from the RecSys Challenge 2015, the data sets *YooChoose* consists of a stream of user clicks on an e-commerce website, collected over six months [Yoo22]. Instead of using the complete data set, many studies have used fractions 1/64 and 1/4 of the sequences as experimental data sets, called *Yoochoose 1/64* and *Yoochoose 1/4*, respectively [Gao+21].

Diginetica The *Diginetica* data set was introduced as part of the 2016 CIKM Cup [DIG16]. It consists of chronologically ordered transaction data and is typically used for session-based recommendations [DIG16; Wu+20a]. For effective personalized recommendations, only transaction data is usually used. Chronological data allow a detailed analysis of user behavior and provide valuable information for developing recommendation systems [DIG16; Wu+20a; Xu+19].

RetailRocket The *RetailRocket* data set was collected from a real e-commerce site. Although it consists of raw data, all values have been hashed for privacy reasons [Ret22; Xu+19]. The data set is divided into three files, one describing behavioral data, another representing item properties, and the third highlighting the category tree [Ret22; Xu+19]. The behavioral data provides information about user activity on the e-commerce site, while the item properties provide details about the products offered. The category tree is used to classify these products [Ret22; Xu+19].

Amazon This data set contains product reviews (ratings, text, helpfulness votes) and metadata (descriptions, category information, price, brand, and image attributes) [McA14; Wu+20a]. The entire data set is segmented into subdata sets by category,

such as Amazon Books, Amazon Instant Video, or Amazon Electronics [McA14; Wu+20a]. These subdata sets can be used to test the performance of collaborative filtering in sequential recommendations for user items [McA14; Wu+20a].

LastFM The LastFM data set includes music listening data from a group of 2,000 users and artist attributes [Ber+11]. Furthermore, this data set provides two types of song-level data: tags and similar songs [Ber+11].

2.5.2. Evaluation metrics

The following list presents the metrics commonly used for sequential and session-based recommendation systems based on GNN.

$P@K$ **(Precision)** is often used to measure prediction accuracy. It represents the proportion of correctly recommended items among the top $\mathcal{K}$ items [Wu+20a; Gao+21]. In the context of recommendation systems, this metric measures the proportion of recommended $\mathcal{K}$ items on which the user clicks [Wu+20a; Gao+21]. A common metric is $P@20(\mathcal{K} = 20)$. This metric as follows [Xu+19; Liu+18]:

$$P@K = \frac{n_{\text{hit}}}{\mathcal{N}} \tag{2.28}$$

In this context, $\mathcal{N}$ represents the number of test data in $\mathcal{X}$, n_{hit} represents the number of cases that have the desired elements in the lists of the highest ranked $\mathcal{K}$ [Liu+18].

$MRR@K$ **(Mean Reciprocal Rank)** This metric reflects the average reciprocal rank of correctly recommended items [Wu+20a; Gao+21]. The reciprocal rank is set to 0 if the rank is greater than $\mathcal{K}$ [Wu+20a; Gao+21]. This metric considers the recommendations' ranking, so a high value MRR reflects the correct recommendations at the top of the ranking list [Wu+20a; Gao+21]. $MRR@K$ is defined as follows [Wu+20a; Xu+19; Liu+18]:

$$MRR@K = \frac{1}{\mathcal{N}} = \sum_{t \in \mathcal{X}} \frac{1}{Rank(t)} \tag{2.29}$$

As can be concluded from the formula, it can generally be stated that $MRR@K$ is a normalized value in the range $[0, 1]$, where the remarks made before the formula apply [Liu+18].

Metrics with negative sampling Because other metrics are based solely on one-way positive feedback from the data [JLJ15]. A previous investigation has proposed using a modified recall variant that also uses negative feedback [JLJ15]. Measure the relative position of the recommended items in a combined set of positive and negative items. To achieve this, each target item t is combined with n random items unknown to the user [JLJ15]. The model then ranks the resulting list of $n+1$ items. A *hit* occurs if t is among the first $\mathcal{K}$ items [JLJ15]. This leads to a statement about the algorithm compared to negative examples. The modified formulas can be described as follows [JLJ15]:

$$\begin{aligned} R@\mathcal{K}^n &= n_{\text{hit}} \\ P@\mathcal{K}^n &= \frac{1}{n+1} \cdot R@\mathcal{K}^n \\ F_1@\mathcal{K}^n &= 2 * \frac{R@\mathcal{K}^n * P@\mathcal{K}^n}{P@\mathcal{K}^n + R@\mathcal{K}^n} \end{aligned} \tag{2.30}$$

In Equation 2.30 $R@\mathcal{K}^n$ is the recall, $P@\mathcal{K}^n$ is the precision and $F_1@\mathcal{K}^n$ is the F-score [JLJ15]. This score is calculated from the precision and recall of the test [JLJ15]. Precision is also known as a positive predictive value, and recall is also known as sensitivity in binary diagnostic classification [SJS06].

2.6. Conclusion

Recommendation systems based on GNNs are a rapidly developing area of research. This section comprehensively reviews recent advances in these systems, focusing on sequential and session-based recommendation systems [Wu+20a; Gao+21]. These systems are particularly efficient due to their ability to account for the sequential nature of user interactions [Wu+20a; Gao+21].

Recognizing the challenges encountered in the development of GNN-based recommendation systems is essential to improve their efficiency and accuracy, thus setting the stage for future deployments [Wu+20a; Gao+21]. Some issues addressed in developing such systems include data scarcity, incorporating contextual factors, and addressing scalability barriers [Wu+20a; Gao+21].

This section lists several commonly used models for sequential and session-based recommendation systems. Table 2.1 and Table 2.2 summarize these main models,

detailing their characteristics and references. These models provide various solutions suitable for different applications and scenarios [Wu+20a; Gao+21].

It is important to note that this overview covers only a portion of the extensive range of potential models [Wu+20a; Gao+21]. Future research may introduce new models and technologies not discussed in this thesis [Wu+20a; Gao+21].

Model	Graph	GNN	Modeling
SURGE Chang et al. [Cha+21]	item-item graph	GAT	RNN
ISSR Liu et al. [Liu+20]	item-item and user-item graph	GCN	RNN
MA-GNN Ma et al. [Ma+19]	item-item graph	GCN	Memory network
DGSR Zhang et al. [Zha+21b]	item-item graph	GAT	RNN
GES-SASRec Zhu, Sun and Chen [ZSC21]	item-item graph	GCN	RNN
RetaGNN Hsu and Li [HL21]	temporal heterogeneous graph	GAT	Self Attention
TGSRec Fan et al. [Fan+21]	temporal user-item graph	GAT	GAT
SGRec Li et al. [Li+21b]	item-item graph	GAT	GAT
GME Xie et al. [Xie+16]	item-item and user-item graph	GAT	GAT
STP-UDGAT Lim et al. [Lim+20]	item-item and user-item graph	GAT	GAT
GPR Chang et al. [Cha+20]	item-item and user-item graph	GCN	GCN

Table 2.1.: Details of GNN models for sequential recommendation, Source: [Gao+21]

Model	Graph	GNN	Modeling
SR-GNN Wu et al. [Wu+19]	directed graph	gated GNN	-
GC-SAN Xu et al. [Xu+19]	directed graph	gated GNN	-
TA-GNN Yu et al. [Yu+20]	directed graph	gated GNN	-
FGNN Qiu et al. [Qiu+19]	directed graph	GAT	-
A-PGNN Zhang et al. [Zha+21a]	directed graph	gated GNN	cross sessions
MGNN-SPred Wang et al. [Wan+20a]	directed-multi-relational item	GraphSAGE	cross sessions
GAG Qiu et al. [Qiu+20]	directed graph	GCN	cross sessions
SGNN-HN Pan et al. [Pan+20]	star graph	gated GNN	additional edges
GCE-GNN Wang et al. [Wan+20b]	directed graph + global graph	GAT	cross sessions
DGTN Zheng et al. [Zhe+20]	directed graph	GCN	cross sessions
DHCN Xia et al. [Xia+20]	hypergraph + line graph	HyperGCN	cross sessions
SHARE Wang et al. [Wan+21a]	session hypergraph	HyperGAT	additional edges
SERec Chen and Wong [CW21]	KG + directed graph	GAT + gated GNN	cross sessions
LESSR Chen and Wong [CW20]	directed graph	GAT	additional edges
CAGE Sheu and Li [SL20]	KG + article-level graph	GCN	cross sessions
MKM-SR Meng, Yang and Xiao [MYX20]	KG + directed graph	gated GNN	cross sessions
COTREC Xia et al. [Xia+21]	item graph + line graph	GCN	cross sessions
DAT-MDI Chen, Guo and Song [CGS21]	directed graph + global graph	GAT	cross sessions
TASRec Zhou et al. [Zho+21]	dynamic graph	GCN	cross sessions

Table 2.2.: Details of GNN models for session-based recommendation, Source: [Gao+21]

3. Temporal Graph Recommendation

This chapter proposes the Temporal Graph Recommendation (TGR) model to address the challenges identified in chapter 1.

In the context of recommendation systems, graphs efficiently represent the relationships between users and items [Wu+20a; Gao+21]. However, traditional GNNs have limitations in effectively capturing the dynamics of these evolving relationships over time [Wu+20a; Gao+21]. In particular, this applies in scenarios where the user's engagement with objects is influenced by various factors, such as the time of day, the context of the interaction, or the fluctuation of user preferences and interests [Wu+20a; Gao+21].

TGR is designed to reflect the dynamic relationships of the user-item interaction graph. For this purpose, TGR uses DGNNs, which are able to continuously update the structure and relationships within the graph to reflect the dynamics of user interactions [Kaz+19]. GNNs are explicitly defined in section 2.2, while an explanation of DGNN can be found in subsection 2.2.5. TGR use a hierarchical structure that enables the integration of session- and sequence-based recommendation systems [Sob21].

This chapter first gives an overview of TGR in section 3.1, then explains the implementation process in section 3.2. The model can be roughly divided into three layers, which are described in section 3.3, section 3.4, and section 3.5. The score function is described in section 3.6. The training strategy is then described in section 3.7, which explains the hyperparameters used in TGR.

3.1. TGR Overview

In this section, the model is presented superficially to provide an overview. First, the three layers of the model are listed, and then the model is presented in its entirety and coherence.

Temporal Graph Recommendation Structural Layer (TGR-STR) An essential foundation of GNN models are the ability to learn information from their neighbors

[Wu+20a; Wu+22]. The idea is that TGR-STR also learns the behavior of users as a whole through the neighborhood of users with items. For this purpose, the neighborhood relations are used for each user $u \in \mathcal{U} \subset \mathcal{X}$ and each item $i \in \mathcal{I} \subset \mathcal{X}$. Consider $\mathcal{I} \in \mathbb{R}^{d^{\mathcal{I}}}, \mathcal{U} \in \mathbb{R}^{d^{\mathcal{U}}}$ for all subsequent discussions. Since $\mathcal{X}$ is defined as a bipartite graph, neighbors $e_{(u,i)} \in \mathcal{E} \subset \mathcal{X}$ to the other node types, for example, neighbors of items $i \in \mathcal{I}$ are always users $u \in \mathcal{U}$. Thus, it is possible that each node learns interaction habits, an item learns which users like to use it, and a user learns which items are frequently interacted with. To address the dynamic nature of user interactions, a DGNN is used for learning in this layer. TGR-STR adapts the proposal suggested in the TGN framework, by Rossi et al. [Ros+20], for deep learning on dynamic graphs. TGR-STR differs from TGN by adapting the approach for recommendation systems. The fundamental concept of using a combination of memory modules and graph-based operators to achieve superior performance while being more efficient than previous approaches remains. TGR-STR is described in more detail in section 3.3.

Temporal Graph Recommendation Session Layer (TGR-SES) An important aspect of the algorithm is the processing of user sessions $s \in \mathcal{S}^u$ as individual session graphs $\mathcal{G}^s$. Each session graph represents the interactions of a user within a session and contains both explicit items that the user has viewed and implicit signals such as the time spent on certain item. This layer focuses on processing these session graphs. Session graphs are processed in several steps. The method *SessionGCN* uses the user's information to personalize the embeddings of the nodes in the session graph. *SessionAttentionNetwork* allows the model to extract the relevant parts of the session graph for a recommendation. Finally, *MultiHeadAttention* is used to combine the embeddings of the nodes in the session graph and obtain a vector representation of the entire session. TGR-SES is described in section 3.4.

Temporal Graph Recommendation Historical Layer (TGR-HIS) In general, sequential and session-based recommendation systems are considered separately. The author proposes TGR-HIS to integrate sequential and session-based recommendation systems into one model. This approach considers a user's session $\mathcal{S}^u$ as historical sessions $\mathcal{S}_h^u$ representing previous interests, provided they are in the past and there is a current session $\mathcal{S}_c^u$. The objective is to use the various sessions $\mathcal{S}^u = \mathcal{S}_h^u \cup \mathcal{S}_c^u$ that a user has accumulated over time to provide the appropriate recommendations for the current session. In TGR-HIS, the historical sessions $\mathcal{S}_h^u$ are combined with

$\mathcal{S}_c^u$ in a *MultiHeadAttention* to better represent the current session in the sequence. Suppose that historical interactions are unavailable, for example, because the user is new to the platform or the system does not store historical interactions. In that case, the purpose of this layer is to respond accordingly. TGR-HIS is described in detail in section 3.5. Therefore, the presented method provides a way to combine sequential and session-based information to generate better recommendations for users. This can help improve the personalization of recommendations and thus increase user satisfaction [Lv+19; Bur02].

The result of these steps are relevant recommendations for the user based on his past interactions. TGR combines graph-based methods, RNNs, linear layers, and attention mechanisms to generate personalized recommendations that consider the dynamic interactions between users and articles.

Figure 3.1 provides a high-level overview of how the TGR model works. Each data set is seen as the input $\mathcal{X}$ of TGR. The data sets are described in detail in chapter 4, consist of user features, item features, and edge features.

First, $\mathcal{X}$ is processed in the TGR-STR layer. For this process, a batch is extracted from the training data (1), where the default proportion of the training data is 60%. TGR-STR uses the current memory $\mathcal{M}^t$ (2) to process the features of the dynamic layer. In the *User - Interaction -Storage* phase (3), information about which user interacted with which item during a given session is stored. This information is crucial for constructing the dynamic graph and the session graphs. The computations of TGR-STR serve a dual purpose: updating the memory $\mathcal{M}^t \rightarrow \mathcal{M}^{t+1}$ (4) and forming the basis for the TGR-SES layer.

The TGR-SES layer constructs an individual session graph $\mathcal{G}^s$ for each user, using the internal memory of the interaction data (6) and the updated memory $\mathcal{M}^{t+1}$ (7). This computation is unique and cannot be classified as batch processing because the data is processed individually. The results of this layer are stored in session memory $\mathcal{M}^{\mathcal{S}}$ (8), overwriting previous entries to ensure that there is only one unique vector per session. The data are then passed to the TGR-HIS layer (9).

If a user has participated in multiple sessions (10) at the time of computation, the corresponding historical embeddings (11) are merged with the current embedding (9). The resulting final embedding (12) is then combined with the item memory $\mathcal{M}^{\mathcal{I}'}$ (13) using a MLP to calculate a score for each item per user (14).

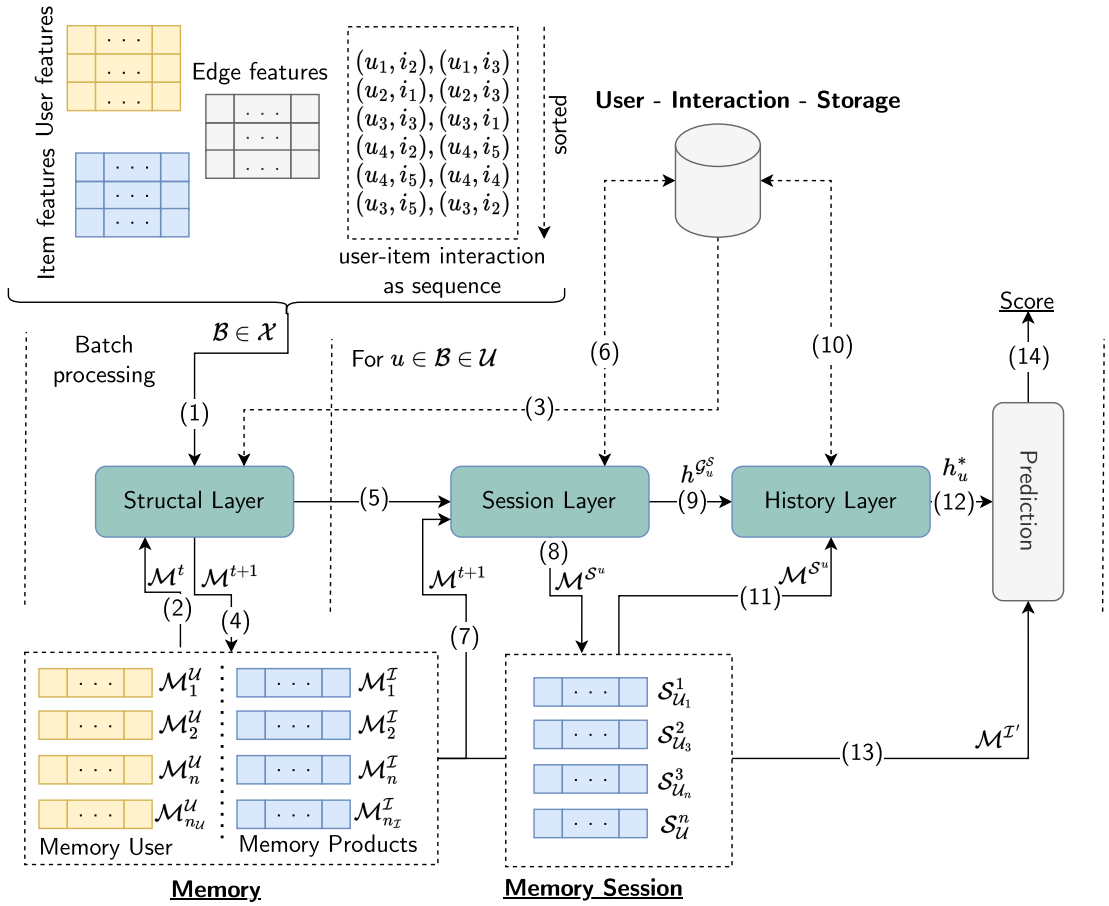


Figure 3.1.: High-level overview of the TGR model with its hierarchical structure

3.2. Implementation

This section provides a concise summary of the implementation process and the general structure of each layer to give a comprehensive overview of TGR. This section refers to algorithm 1 to explain the algorithm. In addition, a graphical overview is presented in Figure 3.2. All the notation in this chapter corresponds to the notation used in the algorithm and the figure.

TGR has a hierarchical structure. A hierarchical structure is recognized when an NN comprises several loosely connected subnetworks organized in layers [MC92]. Each subnetwork is designed to encapsulate specific aspects of the input data [MC92]. Although subnetwork models are a subset of input variables, precise patterns and relationships among variables are established by training the entire network [MC92].

Within TGR, dual iterative processes are used during the training phase, which is significant both in the implementation and in training. The outer loop of the algorithm separates the data set $\mathcal{X}$ into distinct batches $\mathcal{B} \in \mathcal{X}$. The inner loop, called the session loop, processes each user $u \in \mathcal{B}$ independently. This configuration allows the model to scale effectively, which is critical given the high complexity of the data sets and the growing user base during the training phase [MC92; Sob21].

Algorithm 1: TGR Model

Data: $\mathcal{X}$

Result: Score per item (y)

while $\mathcal{B} \in \mathcal{X}$ **do**
  $h^n_{\mathcal{V}} \leftarrow$ *TemporalNeighborEmbedding*$(\mathcal{B}, \mathcal{M}^t)$;
  $\mathcal{M}^{t+1} \leftarrow$ *TemporalNeighborUpdate*$(h^n_{\mathcal{V}}, \mathcal{M}^t)$;
  while $u \in \mathcal{B}$ **do**
    $\mathcal{G}^s \leftarrow$ *BuildSessionGraph*$(u, \mathcal{M}^{t+1})$;
    $h^{\mathcal{G}^s}_{ggc} \leftarrow$ *SessionGCN*$(\mathcal{G}^s)$;
    $h^{\mathcal{G}^s}_{se} \leftarrow$ *SessionAttentionNetwork*$(h^{\mathcal{G}^s}_{ggc})$;
    $h^{\mathcal{G}^s} \leftarrow$ *SessionUserCombination*$(h^{\mathcal{G}^s}_{se}, u)$;
    $\mathcal{M}^{\mathcal{S}^u} \leftarrow$ *SaveSessionEmbeddings*$(h^{\mathcal{G}^s})$;
    if $|\mathcal{M}^{\mathcal{S}^u}| \geq 2$ **then**
      $h^{\mathcal{H}^u}_{ha} \leftarrow$ *HistoricalAttention*$(h^{\mathcal{G}^s}, \mathcal{M}^{\mathcal{S}^u})$;
      $h^*_u \leftarrow$ *HistoricalCombination*$(h^{\mathcal{G}^s}_{se}, h^{\mathcal{H}^u}_{ha})$;
    else
      $h^*_u \leftarrow h^{\mathcal{G}^s}$;
    end
  end
  $\hat{y} \leftarrow$ *Score*$(h^*_u, \mathcal{M}^{\mathcal{I}'})$
end

The training loop of the proposed model TGR, as shown in algorithm 1, processes each batch $\mathcal{B} \in \mathcal{X}$ by embedding a temporal neighbor using the function *TemporalNeighborEmbedding* with the current batch $\mathcal{B}$ and the current memory $\mathcal{M}^t$ ($h^n_{\mathcal{V}} \leftarrow$ *TemporalNeighborEmbedding*$(\mathcal{B}, \mathcal{M}^t)$). The hyperparameter BATCH-SIZE controls the batch length, indirectly determining the number of batches possible. This neighborhood is then updated using the *TemporalNeighborUpdate* function, which computes the context

of the users' and items' neighborhoods to gain structural information about the data set. The function updates the model, where $\mathcal{M}^{t+1} = \textit{TemporalNeighborUpdate}(h^n_{\mathcal{V}}, \mathcal{M}^t)$. A more detailed description of this function can be found in section 3.3.

For each user $u \in \mathcal{B}$ in the proposed TGR-SES model, a session graph $\mathcal{G}^s$ is generated by applying the *BuildSessionGraph* function to the current user u and the updated memory $\mathcal{M}^{t+1}$. This graph represents the user's interactions within the current session and is the foundation for subsequent processing. The $h^{\mathcal{G}^s}_{ggc}$ is then passed through *SessionGCN* and *SessionAttentionNetwork*, which consider the graph-based context of the user's interactions and provide a weighted consideration of relevant interactions within the session, respectively. The resulting hidden vector $h^{\mathcal{G}^s}$ is generated using the function *SessionUserCombination* with $h^{\mathcal{G}^s}_{se}$ and u.

After processing TGR-STR, the TGR-HIS layer is applied to combine the historical sessions represented by $\mathcal{M}^{\mathcal{S}^u}$ with the current session $\mathcal{S}^u_c$. If there are no historical sessions for the user, this layer can be skipped, as indicated in the algorithm. The final embedding of the session $h^{\mathcal{G}^s}$ is combined with $\mathcal{M}^{\mathcal{S}^u}$ using *HistoricalAttention* to access information from previous sessions and improve predictions, returning the hidden vector $h^{\mathcal{G}^s}_{se}$. Then *HistoricalCombination* is applied, using $h^{\mathcal{H}^u}_{ha}$ and $h^{\mathcal{G}^s}_{se}$ to calculate h^*_u. In TGR-HIS, the current session is weighted more heavily to account for short-term changes in the user's interests.

The score $\hat{y} = \text{score}(h^*_u, \mathcal{M}^{\mathcal{I}'})$ is calculated, which synthesizes the previously calculated scores. This score represents the prediction made by the model of the user's interest in the items in $\mathcal{M}^{\mathcal{I}'}$. The proposed model utilizes temporal and graph-based information to model user behavior and provide personalized recommendations. More details on the architecture and implementation of the model can be found in the following sections.

3.2.1. Recurring functions

In this thesis, various mathematical methods and models are used, as described in the beginning of this section. These functions assume a central role, which serve as building blocks for more complex expressions and equations. This subsection gives an overview of these basic functions and describes their properties, which will be of central importance in the following phases of the thesis.

It is important to emphasize that these functions used in this thesis should not be considered isolated entities. Rather, they are integral parts of a larger model and contribute to the formation of more complex expressions and equations, which will be

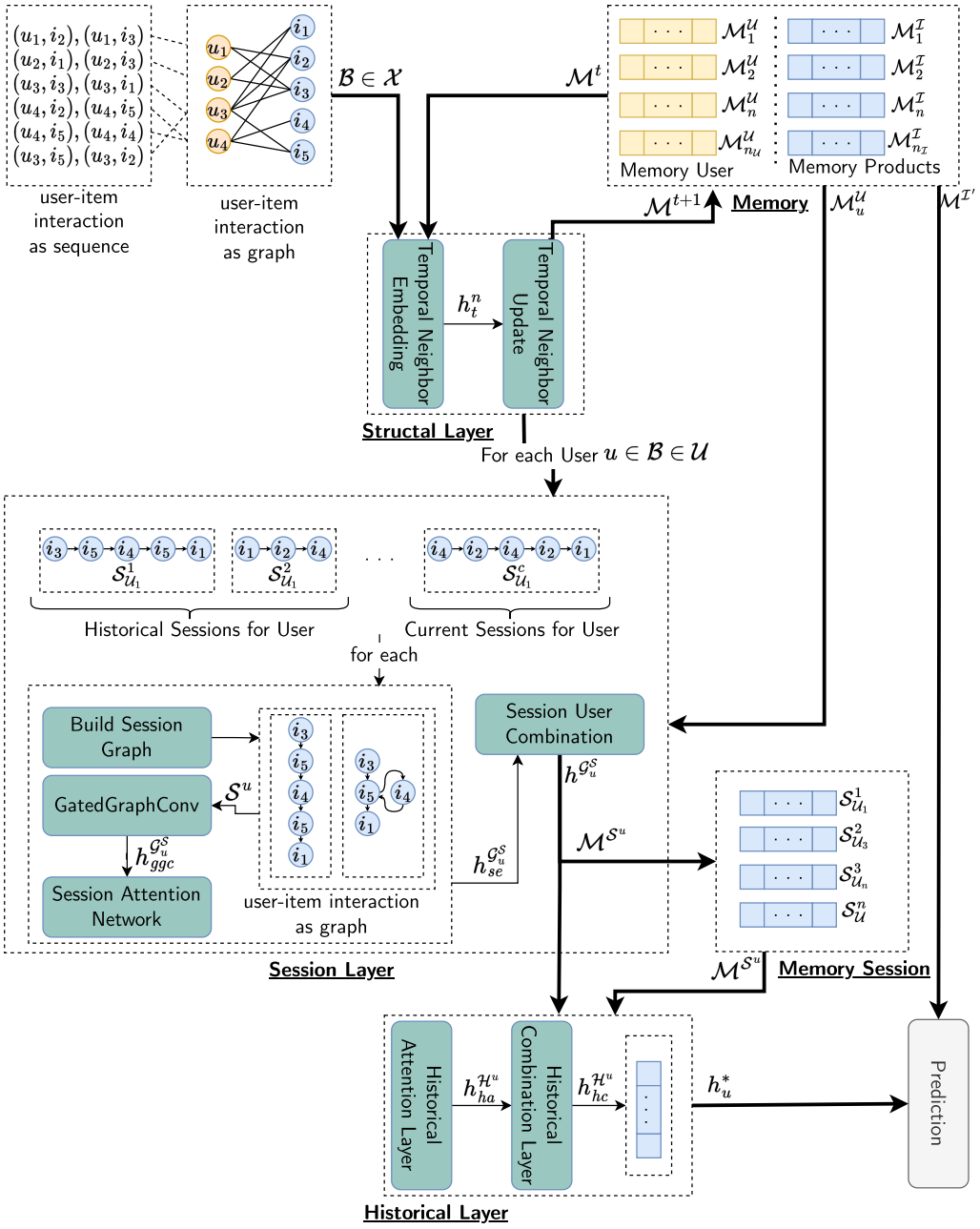


Figure 3.2.: Overview of the TGR model with its hierarchical structure

explored in later sections of the thesis. These functions have certain properties and behaviors that allow them to be applied and integrated into more complex models.

Linear layer

The linear layer is one of the most fundamental and commonly used components in NN [Jun22]. This layer transforms the input data $\mathcal{X}_{lin}$ into the output data $\mathcal{Y}$ using a linear transformation [Jun22; Moc+16].

Mathematically, the Linear($\mathcal{X}$) layer can be described as shown in Equation 3.1.

$$\text{Linear}(\mathcal{X}) = \mathcal{Y}_l = \mathcal{W}\mathcal{X}_{lin} + b \tag{3.1}$$

Where $\mathcal{X}_{lin}$ is the input matrix, here $\mathcal{X}_{lin} \in \mathbb{R}^{n \times d}$, where n is the number of observations, and d is the number of features per observation [Jun22; Moc+16]. $\mathcal{Y}_l \in \mathbb{R}^{n \times m}$ is the output matrix, $\mathcal{W} \in \mathbb{R}^{d \times m}$ is the weight matrix of the layer and $b \in \mathbb{R}^{1 \times m}$ is the bias vector [Jun22; Moc+16].

The linear layer allows for a fundamental transformation of the input data, which is optimized by adjusting the weights and biases during the training process [Jun22; Moc+16].

MergeMLP

In this thesis, a function is used to concatenate two vectors and then reduce their dimensionality to a given dimension. This function uses Multilayer Perceptrons (MLPs) to assign different weights to the information contained in the vectors. The function is denoted as MergeMLP(x_1, x_2, d_3), where it uses ELU defined in Equation 2.17 and the weight matrices $\mathcal{W}_1$ and $\mathcal{W}_2$ as parameters. During training, these parameters are optimized to capture and model complex relationships between the vectors to be concatenated.

$$\text{MergeMLP}(x_1, x_2, d_3) = \mathcal{W}_2 \,\text{ELU}(\mathcal{W}_1[x_1; x_2])) \tag{3.2}$$

In Equation 3.2, the first vector is denoted as $x_1 \in \mathbb{R}^{d_1}$ and the second vector is denoted as $x_2 \in \mathbb{R}^{d_2}$. The notation $[x_1; x_2]$ represents the concatenation of the vectors, which means that $[x_1; x_2] \in \mathbb{R}^{(d_1+d_2)}$ is true. The output is represented as $x \in \mathbb{R}^{d_3}$. The corresponding weight matrices $\mathcal{W}_1, \mathcal{W}_2$ determine the weights of the respective input vectors and thus have the corresponding dimensions ($\mathcal{W}_1 \in \mathbb{R}^{d_3 \times (d_1+d_2)}, \mathcal{W}_2 \in \mathbb{R}^{d_3 \times d_3}$).

3.3. TGR Structural layer

The first layer of the hierarchically structured model TGR is Temporal Graph Recommendation Structural Layer (TGR-STR), which serves an essential function in this recommendation model.

To illustrate the importance of the first layer in hierarchical networks, consider image classification with NNs. For example, the first layer can detect important features such as edges, curves, and textures relevant to the final classification. Similarly, the first layer in TGR detects important features, such as the data structure [Eln+21]. The underlying principles of the model are detailed in subsection 3.3.1. This is followed by a description of the model implementation strategy in subsection 3.3.2. In addition, the formal definition of the model is presented in this section.

3.3.1. Foundation

The previous literature discusses encoder-decoder pairs for DGNN. An encoder maps a dynamic graph to the node embeddings, while the decoder uses these embeddings to make task-specific predictions [Kaz+19; Zhu+22].

Rossi et al. [Ros+20] proposes TGN, an encoder-decoder architecture, as a framework to analyze and develop DGNN. The encoder presented in TGN is a method for continuous-time dynamic graphs that achieves state-of-the-art performance in experiments. The encoder generates the embedding of the nodes $\mathcal{Z}(t) = (z_1(t), ..., z_{n_t}(t))$ for each time t [Ros+20]. TGN use an MLP decoder mapping from the concatenation of two node embeddings to the probability of the edge [Ros+20].

This approach is suitable for recommender systems because using a dynamic graph allows to capture the evolving relationships between nodes over time. Changes between nodes can be tracked by computing the embedding vector for each node at each time point. Consequently, these embedding vectors can compute recommendations [Cha+21; Gao+21; Wan+21b].

Using the robust foundational work of TGN within the domain of DGNN [Ros+20], TGN forms the foundation of TGR-STR. The core of TGN is the memory $\mathcal{M}$, which enables the model to store long-term dependencies for each node in the graph [Ros+20]. This is achieved by storing a vector for each node that is updated after an event, such as a user interacting with an element, occurs. The purpose of the memory $\mathcal{M}$ is to present the history of each node in a condensed form [Ros+20].

TGN expects homogeneous graphs for computations [Ros+20]. On the contrary,

recommendation systems inherently rely on heterogeneous graphs [Gao+21]. This is essential for properly representing relationships between different types of entities, such as users and items. A heterogeneous structure also allows the consideration of different types of information, including age, gender, purchase history, and social relationships [Gao+21]. This structure increases the accuracy and relevance of recommendations [Gao+21]. For these reasons, TGR adopts the methodology used by TGN to accommodate heterogeneous graph structures. In this thesis, a heterogeneous bipartite graph is used for this purpose. Future research could, among other things, explore the benefits of including social relations in this model, extending the heterogeneous bipartite graph structure. To improve training efficiency, TGN uses batch processing. Using batch processing may result in nodes appearing more frequently in the same batch [Ros+20]. These are grouped in TGN before memory updates for a better runtime [Ros+20]. For example, if the item $i_3 \in \mathcal{I}$ occurs nine times in the current batch, it will negatively affect the runtime of the training process to perform a memory update for each of these nodes, so TGN suggests that aggregating the nodes does not negatively affect runtime [Ros+20]. In contrast, TGR-STR avoids this aggregation approach because evidence from other research suggests that each interaction is invaluable for an accurate recommendation [Cha+21; Wu+20a; Gao+21; Wan+21b]. Still, batch processing is used in TGR-STR.

In addition, the processing of the nodes of TGN and TGR-STR are different. A detailed explanation of how TGR-STR works is given in the following section.

3.3.2. Implementation

TGR-STR is described graphically in Figure 3.3 and algorithmically in algorithm 2.

$\mathcal{X} = (\mathcal{V}, \mathcal{E})$ is a heterogeneous graph in which each node is either a user or an item. Therefore, $u \in \mathcal{U} \subset \mathcal{V}, i \in \mathcal{I} \subset \mathcal{V}$. According to the definition of $e_{(u,i)} \in \mathcal{E}$, an edge cannot occur between nodes of the same type. Furthermore, due to the background of DGNN, each node can occur in multiple versions. This implies that the vectors describing the nodes do not have to be static. They can change over time. For example, i_3 can have a different vector at time t than at time $t-1$. However, the features of $\mathcal{E}$ are static and do not change over time. Only new edges can be added to the graph; existing edges are not updated or deleted. Therefore the graph cannot be considered fully dynamic, as described in subsection 2.2.5, since no edges are deleted or updated, which cannot happen in recommendation systems [Gao+21]. The algorithm processes the input data and updates the memory $\mathcal{M}$ that contains the representation of the users or items.

Algorithm 2: TGR Structural Layer

Data: Raw data, user or item as $\mathcal{V} = \mathcal{U} \wedge \mathcal{I}$

Result: Updatet Storage ($\mathcal{B}$)

while $\mathcal{B} \in \mathcal{X}$ **do**

$\mathcal{V}^t \leftarrow \mathcal{V} + \mathcal{M}^t$;

$\mathcal{N} \leftarrow \mathit{FindNeighbors}(\mathcal{M}^t))$;

$\mathcal{T}_\mathcal{V} \leftarrow \mathit{EncodeTime}(t \in \mathcal{E})$;

$\mathcal{T}_\mathcal{N} \leftarrow \mathit{EncodeTime}(t \in \mathcal{N})$;

$\mathcal{Q}^t \leftarrow [\mathcal{V}^t; \mathcal{T}_\mathcal{V}]$;

$\mathcal{K}^t, \mathcal{V}^t \leftarrow [\mathcal{N}; \mathcal{N}^\mathcal{E}; \mathcal{T}_\mathcal{N}]$;

$h_t^n \leftarrow \mathit{MultiHeadAttention}(\mathcal{Q}^t, \mathcal{K}^t, \mathcal{V}^t)$;

$h_\mathcal{V}^{n\prime} \leftarrow \text{MergeMLP}(\mathcal{V}, h_t^n, d^\mathcal{V})$;

$\mathcal{M}^{t+1} \leftarrow \text{GRU}(h_\mathcal{V}^{n\prime}, \mathcal{M}^t)$;

end

Since all layers of TGR are based on batch processing, a limited number of records are extracted from $\mathcal{E}$ such that $\mathcal{B} \in \mathcal{E}, |\mathcal{B}| = \text{BATCH-SIZE}$. For each interaction, $e_{(u,i)} \in \mathcal{E}$, a negative sample is incorporated [Arm+19]. Negative sampling is a technique used to reduce computation time and memory requirements when training NNs [Arm+19]. The amount of training data can be reduced without compromising the quality of the embeddings by using negative examples [Arm+19]. This results in more effective use of the training data, faster training time, and higher memory efficiency [Arm+19]. The author proposes several methods to generate negative samples that can be selected by the hyperparameter NEIGHBOR-SAMPLER.

The *popular neighbor problem* can arise with negative sampling. This problem refers to a phenomenon in the context of negative sampling that can occur when learning NNs [Arm+19]. It states that frequently occurring nodes in the training data set generate a disproportionately high number of negative examples, which can lead to the model overfitting to these frequent nodes instead of achieving good general testability on unknown data [Arm+19]. To avoid this problem, there are several approaches, such as weighting the negative examples according to their frequency [Arm+19].

Random Negative Sampling (RNS) This is a basic negative sampling algorithm, which aims to treat each item in the sample pool equally and sample them with equal probability to generate a negative sample [Ren+12; Yan+20]. A pseudo-random

number generator selects a random item from all existing items before the current interaction. This process ensures that each item in the sample pool has an equal chance of being selected as a negative example [Ren+12; Yan+20]. However, when using RNS, the sample pool size must be well chosen, as it can affect the quality of the negative example generated [Ren+12]. In this thesis, due to the important time aspect, only random items that are already known to the system are selected, but if there are not enough random items in the early batches that are already known to the system, the pool of all possible items is made up of the pool of all possible items.

Similarity-based Negative Sampling (SBNS) Using an item that is either highly similar or highly dissimilar to train the model on the difference between similar items offers another way to generate negative examples [Yao+22]. This method is called SBNS in this thesis. The similarity calculation is performed before training the model to ensure high efficiency [Yao+22; Ren+12]. A hierarchical approximation structure for non-metric spaces is used to find similar objects. This structure is specifically optimized for cosine similarity and significantly speeds up the search for similar objects. Cosine similarity measures the similarity between two vectors in a multidimensional space. Consequently, each element $i \in \mathcal{I}$ is assigned another element that is as similar or dissimilar as possible to the vector of i, depending on the hyperparameter. Using similar items as negative examples helps improve the ability of the model to distinguish between similar items and increase the overall accuracy of the model [Ren+12].

In Figure 3.3, the user and item computations are shown together, but the user and item embeddings are computed separately.

The representation of the users and items contained in the batches is taken from the memory $\mathcal{M}^t$ at time t ($\mathcal{M}^{\mathcal{U}^t} \cap \mathcal{M}^{\mathcal{I}^t} \in \mathcal{M}^t$). If the nodes are not seen by the model, they are set to zero. The representations of users or items contained in the batches are combined with those already in storage. This results in an updated representation of users $\mathcal{U}^t = \mathcal{U} + \mathcal{M}^{\mathcal{U}^t}$. This representation contains all the information about the node for which $t < t^c$ applies. Similarly, the item nodes are also updated: $\mathcal{I}^t = \mathcal{I} + \mathcal{M}^{\mathcal{I}^t}$.

TGR-STR uses the concept of *Temporal Graph Attention* from Xu et al. [Xu+20a] to compute node embeddings. This proved to be a powerful tool for modeling the temporal evolution of nodes in a graph. Using a series of L graph attention layers, information from the L-hop temporal neighborhood of a node can be aggregated to

calculate its embedding [Xu+20a; Ros+20]. In this thesis, 1-hop ($L = 1$) graph attention is chosen for the advantage of superior running time [Xu+20a; Ros+20]. Whether increasing the layers can positively affect the runtime of this model is not further explored in this thesis and is left open for future research. Previous research suggests that limiting it to 1-hop does not negatively impact the results [Xu+20a; Ros+20]. Since the temporal neighborhood of a node is used to calculate node embeddings, it is important to identify these neighborhood nodes. This neighborhood can be considered as an interaction ($e_{(u,i)} \in \mathcal{E}$). Therefore, k interactions with items i are identified per u and k users u per item i. It should be noted that k is the hyperparameter NUM-NEIGHBORS. The author suggests that if a user has fewer than $k-1$ interactions in $\mathcal{M}^{\mathcal{U}}$ ($|e^u| \in \mathcal{E}^{\mathcal{U}} < k-1$), the calculations are performed with masks. This can be formally described as $\mathcal{N}_{\mathcal{U}} = \mathit{FindNeighbors}(\mathcal{M}^t), \mathcal{N}_{\mathcal{I}} = \mathit{FindNeighbors}(\mathcal{M}^t)$.

The time of the edges $t \in \mathcal{E}$ will be encoded by the proposal of TGN [Ros+20; Xu+20a]. $\mathcal{T}_{\mathcal{U}} = \mathit{EncodeTime}(t), \mathcal{T}_{\mathcal{I}} = \mathit{EncodeTime}(t)$. In addition, the time of the memory is encoded. The time of the memories reflects the last interaction of this node. $\mathcal{T}_{\mathcal{N}_{\mathcal{U}}} = \mathit{EncodeTime}(t \in \mathcal{N}_{\mathcal{U}}), \mathcal{T}_{\mathcal{N}_{\mathcal{I}}} = \mathit{EncodeTime}(t \in \mathcal{N}_{\mathcal{I}})$ For this purpose, the following formula is used, defined by the equation Equation 3.3.

$$b = 1/10 * *\mathit{linspace}(0, 9, \mathit{dimension}) \tag{3.3}$$

$$\mathcal{T}_{\mathcal{V}} = \cos(\text{Linear}(t)) \tag{3.4}$$

To illustrate *EncodeTime*, the dimension is set to 10, and the resulting array for b would be $[1, 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001, 0.000000001]$. Linear represents a linear layer as defined in subsubsection 3.2.1 and b the corresponding bias term.

Since the structure of the graph changes at each iteration ($t \to t+1$), the model must be able to respond quickly to these changes. For this reason, hidden vectors ($h^n_{t_{\mathcal{U}}}, h^n_{t_{\mathcal{I}}}$) are computed to understand the relationships between the updated node embeddings (u^{t+1}, i^{t+1}) and the existing node embeddings (u^t, i^t). This can be described formally as follows:

$$h^n_{t_{\mathcal{U}}} = \mathit{MultiHeadAttention}(\mathcal{Q}^t_{\mathcal{U}}, \mathcal{K}^t_{\mathcal{I}}, \mathcal{V}^t_{\mathcal{U}}),$$
$$h^n_{t_{\mathcal{I}}} = \mathit{MultiHeadAttention}(\mathcal{Q}^t_{\mathcal{I}}, \mathcal{K}^t_{\mathcal{I}}, \mathcal{V}^t_{\mathcal{I}})$$

MultiHeadAttention is described in detail in subsection 2.2.7. For this purpose, the use of Multi-Head Attention Network (MATN) is a suitable option, as it has already

been proven in the context of DGNNs in previous work [Ros+20]. MATN allows the model to use multiple attention mechanisms directed at different parts of the graphIn this scenario, as described in subsection 2.2.7. This allows the model to respond quickly to changes in the structure of the graph and to understand the relationships between the new and existing nodes [Ros+20; Kaz+19; Zhu+22]. Specifically, $\mathcal{Q}_{\mathcal{U}}^t = [\mathcal{U}^t; \mathcal{T}_{\mathcal{U}}], \mathcal{K}_{\mathcal{U}}^t = \mathcal{V}_{\mathcal{U}}^t = [\mathcal{N}_{\mathcal{U}}; \mathcal{N}_{\mathcal{U}}^{\mathcal{E}}; \mathcal{T}_{\mathcal{N}_{\mathcal{U}}}]$ correspond to the input, key, and value in MATN, respectively. $\mathcal{N}_{\mathcal{U}}^{\mathcal{E}}$ stands for the attributes of the edges between the neighbors and the current node. The same applies accordingly to the calculation of $\mathcal{Q}_{\mathcal{I}}^t = [\mathcal{I}^t; \mathcal{T}_{\mathcal{I}}], \mathcal{K}_{\mathcal{I}}^t = \mathcal{V}_{\mathcal{I}}^t = [\mathcal{N}_{\mathcal{I}}; \mathcal{N}_{\mathcal{I}}^{\mathcal{E}}; \mathcal{T}_{\mathcal{N}_{\mathcal{I}}}]$.

To focus on the features of the current time slice, *MergeMLP* is used. This method is already defined in subsubsection 3.2.1. Here, the embedding $h_{t_{\mathcal{U}}}^n, h_{t_{\mathcal{I}}}^n$ is combined with the representation of the current time step $(t+1)$, and then the dimension is reduced. $h_{\mathcal{U}}^{n\prime} = \text{MergeMLP}(\mathcal{U}, h_t^n, d^{\mathcal{U}}), h_{\mathcal{I}}^{n\prime} = \text{MergeMLP}(\mathcal{I}, h_t^n, d^{\mathcal{I}})$. Using the new embedding $h_{\mathcal{U}}^{n\prime}$ and $h_{\mathcal{I}}^{n\prime}$ allows the model to highlight the most important features of $\mathcal{U}$ and $\mathcal{I}$ nodes. To update the memory in this thesis, GRU is used:

$$\begin{aligned} \mathcal{M}^{\mathcal{U}^{t+1}} &= \text{GRU}(h_{\mathcal{U}}^{n\prime}, \mathcal{M}^{\mathcal{U}^t}), \\ \mathcal{M}^{\mathcal{I}^{t+1}} &= \text{GRU}(h_{\mathcal{I}}^{n\prime}, \mathcal{M}^{\mathcal{I}^t}), \\ \mathcal{M}^{t+1} &= \mathcal{M}^{\mathcal{U}^{t+1}} \cup \mathcal{M}^{\mathcal{I}^{t+1}} \end{aligned}$$

The combined usage of MATN, *MergeMLP*, and GRU can significantly improve the accuracy of the model, as it allows the model to respond quickly and effectively to changes in the structure of the graph and understand the relationships between nodes. Subsequently, the nodes in $\mathcal{M}^{t+1}$ have all the most important information about the neighborhood of the nodes, as well as any new features of the nodes themselves. This allows subsequent layers to use these embeddings, providing a significant performance benefit in computation time since the memory only needs to be computed once and then incrementally updated.

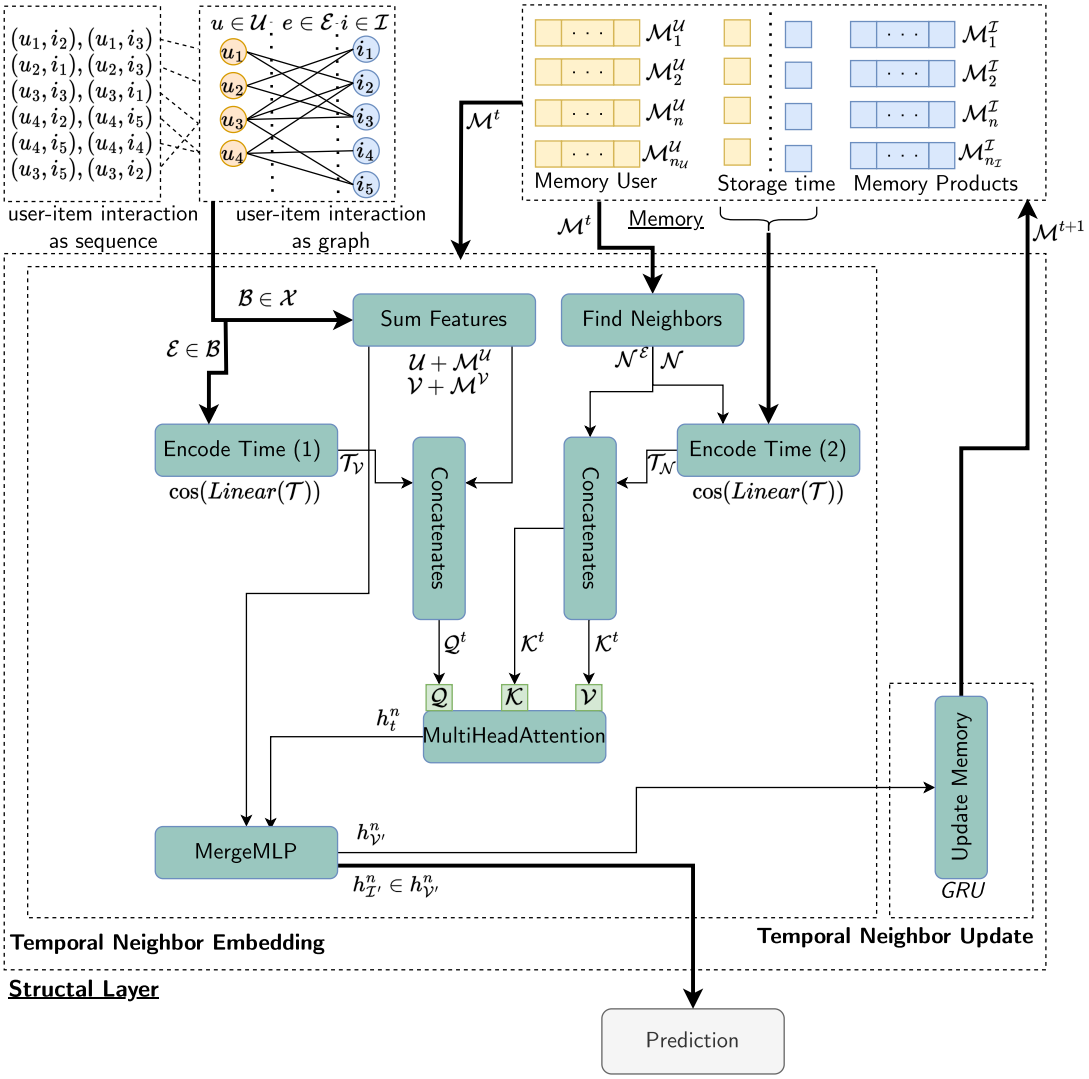


Figure 3.3.: The TGR-structural layer is an algorithm part of a recommendation system based on heterogeneous graphs. The algorithm traverses batches of raw data and updates the storage containing the representations of users and items by considering the relationships between the users, items, and events in the data. Functions such as MATN, *MergeMLP*, and GRU are applied to update and improve the representations of users and items in the memory.

3.4. TGR Session Layer

In this section, the second layer of the hierarchically structured model TGR is discussed in detail. This layer, named Temporal Graph Recommendation Structural Layer (TGR-STR), is specifically designed to capture and analyze user interaction patterns and utilizes session graphs to represent users' interactions with different elements over a period of time. First, in subsection 3.4.1, the motivation is discussed. Then, in subsection 3.4.2, the implementation strategy for the model is outlined. This section also presents a formal definition of the model.

3.4.1. Foundation

The calculations performed on this layer utilize the results derived from the TGR-STR layer as presented in section 3.3. Consequently, in this layer, the variable $\mathcal{M}^{t+1}$ is expected to represent all information known to the model up to time $(t+1) \leq t^c$. Where t^c corresponds to the time of the last element of the current batch. It is important to emphasize that these nodes now include neighborhood information and edge features. The purpose of this layer is to use this information by processing it with a session graph. A session graph is generally described in section 2.4.

In summary, a session graph $\mathcal{G}^s$ can be characterized as a directed graph, where $\mathcal{G}^s = (\mathcal{V}^s, \mathcal{E}^s)$. In this definition, $\mathcal{V}^s$ denotes the set of all nodes corresponding to each interaction in a session, and $\mathcal{E}^s$ denotes the set of all edges representing the sequential relationship between these interactions [Gao+21; Wu+19; Wan+20b].

The transition from one interaction to the next is denoted by a directed edge, which can have different weights. For example, the time spent on a particular action can be assigned to these edges [Gao+21; Wu+19]. A session graph allows analyzing the relationships between different items within a session, helping to identify patterns and trends in user interactions [Gao+21; Wu+19].

TGR-SES is applied to current and previous sessions. However, it is important to note that all user sessions are considered independently. The method proposed here refines the calculation of individual sessions and incorporates additional information, such as time spent on items and CATEGORY-SHIFT, into the process. These aspects are discussed in more detail in the following subsections.

3.4.2. Implementation

TGR-SES processes each user $u \in \mathcal{B}$ independently. For this purpose, the storage data $\mathcal{M}^{t+1}$ are used. The items $\mathcal{M}^{\mathcal{I}} \subset \mathcal{M}^{t+1}$ are reduced to a smaller dimension. Here, $\mathcal{M}^{\mathcal{I}'} = \mathcal{W}_1 \mathcal{M}^{\mathcal{I}} + b_1$ applies. Here, the items $\mathcal{M}^{\mathcal{I}} \in \mathbb{R}^{d^{\mathcal{I}}}$ are accordingly represented in a high-dimensional space. The learnable weighting matrix $\mathcal{W}_1 \in \mathbb{R}^{(d^{\mathcal{I}'} \times d^{\mathcal{I}})}$ and the bias term $b \in \mathbb{R}^{d^{\mathcal{I}}}$ are also included.

Algorithm 3: TGR Session Layer

Data: Storage $\mathcal{M}$

Result: Session Storage Embedding ($\mathcal{M}^{\mathcal{S}}$)

Algorithm: TGR Session Layer

while $u \in \mathcal{B}$ **do**

- $\mathcal{M}^{\mathcal{U}} \leftarrow \mathcal{M}^{\mathcal{U}} \in \mathcal{M}$;
- $\mathcal{M}^{\mathcal{I}} \leftarrow \mathcal{M}^{\mathcal{I}} \in \mathcal{M}^{\mathcal{U}} \cap \mathcal{M}$;
- $\Delta\mathcal{T} \leftarrow \mathit{EncodedEdgeTimes}(t \in \mathcal{M})$;
- $\mathcal{G}^s \leftarrow \mathit{BuildSessionGraph}(u, \mathcal{M}_s^{\mathcal{I}'}, \Delta\mathcal{T})$;
- $h_{ggc}^{\mathcal{G}^s} \leftarrow \mathit{SessionGCN}(\mathcal{G}^s)$;
- $h_{se}^{\mathcal{G}^s} \leftarrow \mathit{SessionAttentionNetwork}(h_{ggc}^{\mathcal{G}^s})$;
- $h^{\mathcal{G}^s} \leftarrow \mathit{SessionUserCombination}(h_{se}^{\mathcal{G}^s}, h_{\mathcal{U}}^{n\prime})$;
- $\mathcal{M}^{\mathcal{S}^u} \leftarrow \mathit{SaveSessionEmbeddings}(h^{\mathcal{G}^s})$;

end

The result of this calculation, $\mathcal{M}^{\mathcal{I}'} \in \mathbb{R}^{d^{\mathcal{I}'}}$ corresponds to the reduced representation of the data in a low-dimensional space. Here, $d^{\mathcal{I}}$ and $d^{\mathcal{I}'}$ represent the dimensions of the high-dimensional and low-dimensional space, respectively. While d is determined by the memory dimension $\mathcal{M}^{\mathcal{I}}$ of TGR-STR, $d^{\mathcal{I}'}$ is controlled by the hyperparameter SESS-TERM-HIDDEN-SIZE. The dimensional reduction technique used here is commonly used in recommendation engineering research to reduce the complexity and size of the data set [Gao+21]. This can improve the performance of the recommendation system and prevent overfitting. It should be noted that determining the reduced dimensions $d^{\mathcal{I}'}$ can have a high impact.

The session graph is described as a directed graph $\mathcal{G}^s = (\mathcal{M}_s^{\mathcal{I}'}, \mathcal{E}^s)$, where $\mathcal{M}_s^{\mathcal{I}'} \in \mathcal{M}^{\mathcal{I}'}$ represents all updated representations of items with which a user u has interacted $\mathcal{E}^s$ in the current session $s \in \mathcal{S}^u \subset \mathcal{S}$. Here $\mathcal{M}_s^{\mathcal{I}'}$ is the general definition of $\mathcal{V}^s$, adapted for TGR-SES. Each node in $\mathcal{G}^s$ represents a click of a particular user. These nodes are sorted

chronologically in the session graph. The dwell time of a session $\mathcal{G}^s$ between two nodes $i_t \in \mathcal{M}_s^{\mathcal{I}'}$ and $i_{t+1} \in \mathcal{M}_s^{\mathcal{I}'}$ can be expressed as $\Delta t = t^{i_{t+1}} - t^{i_t}$, where t^{i_t} denotes the interaction time with i_t. The definition of dwell time is the difference between the time of the subsequent node and the current node. Consequently, $\Delta\mathcal{T}$ contains all dwell times within session s and is referred to *EncodedEdgeTimes*. This information can then be used to construct a session graph $\mathcal{G}^s$ in which an edge $e_{(i_t, i_{(t+1)})} \in \mathcal{E}^s$ with the corresponding weight Δt is created between the corresponding nodes. This procedure is repeated for each edge within the session s. The entire process of creating the session graph can be summarized with the notation per user: $\mathcal{G}^s = \mathit{BuildSessionGraph}(u, \mathcal{M}_s^{\mathcal{I}'}, \Delta\mathcal{T})$. The embedding of the session graph $\mathcal{G}^s$ is calculated using Gated Graph Sequence Neural Networks (GGS-NN). This can be represented by $h_{ggc}^{\mathcal{G}^s} = \mathit{SessionGCN}(\mathcal{G}^s)$. GGS-NN is referred to as *SessionGCN* and can capture complex patterns and relationships in the data [Li+15]. In addition, adaptive information processing allows GGS-NN to select and consider relevant temporal information from neighboring interactions [Li+15]. This is particularly important because not all previous interactions may be equally relevant [Wu+19]. In general, GGS-NN is chosen due to its strong performance and its ability to effectively model temporal session data [Li+15]. Li et al. [Li+15] introduces GGS-NN, an extension of GNN that generates sequences and is explained in Equation 3.5.

$$h_i^{(0)} = [x_i; \mathbf{0}] \tag{3.5}$$

$$m_i^{(l+1)} = \sum_{j \in \mathcal{N}_i} e_{i,j} \cdot \mathcal{W}_3 \cdot h_j^{(l)} \tag{3.6}$$

$$h_i^{(l+1)} = \mathrm{GRU}(m_i^{(l+1)}, h_i^{(l)}) \tag{3.7}$$

The initialization of the node state vectors is achieved by transferring the node annotations into the primary components of the hidden state and filling the remaining part with zeros. This process is represented by the equation $h_i^{(0)} = [x_i; \mathbf{0}]$ [Li+15]. $h_i^{(0)}$ denotes the initial state vector of node i, x_i denotes the annotation of node i [Li+15]. Edge activations are computed bidirectionally, as given by the equation $mi^{(l+1)} = \sum_{j \in \mathcal{N}_i} e_{i,j} \cdot \mathcal{W}_3 \cdot h_j^{(l)}$ [Li+15]. In this equation, $mi^{(l+1)}$ represents the message vector of node i at step $l+1$, $\mathcal{N}_i$ denotes the neighbors of node i, $e_{i,j}$ represents the edge from node j to node i, $\mathcal{W}_3$ is the weight matrix and $h_j^{(l)}$ is the state vector of node j at step l [Li+15]. The hidden state is updated by integrating information from other nodes and the previous time step, as represented by equation $h_i^{(l+1)} = \mathrm{GRU}(m_i^{(l+1)}, h_i^{(l)})$ [Li+15]. Here $h_i^{(l+1)}$ is the state vector of node i at step $l+1$ [Li+15]. $h_{ggc}^{\mathcal{G}^s}$ is utilized to compute

a session embedding, represented by the equation:

$$h_{se}^{\mathcal{G}^s} = \textit{SessionAttentionNetwork}(h_{ggc}^{\mathcal{G}^s})$$

SessionAttentionNetwork is derived from the Session-based Recommendation with Graph Neural Networks (SR-GNN) model conceived by Wu et al. [Wu+19]. Consequently, the terminology and calculations in this adaptation may not directly match those found in the original work of Wu et al. [Wu+19]. The primary input to *SessionAttentionNetwork* is the embedding of a user session $h_{ggc}^{\mathcal{G}^s}$, a previously computed hidden vector. The context in which this function is used differs from Wu et al. [Wu+19]. However, calculating α_i and s_g are used in an adapted format [Wu+19].

$$\alpha_i = \mathcal{W}_6\sigma(\mathcal{W}_4 v_{s,\mathcal{T}} + \mathcal{W}_5 h_{ggc}^{\mathcal{G}^s}) \tag{3.8}$$

$$s_g = \sum_{i=0}^{n} \alpha_i v_{s,i} \tag{3.9}$$

$$s_h = \text{MergeMLP}(s_g, s_c, d^{\mathcal{I}'}) \tag{3.10}$$

The purpose of the computation in Equation 3.8 is to generate local and global embeddings. These two types of embeddings represent different facets of the session data and are ultimately merged into a hybrid embedding that encapsulates the entire session. This corresponds to the principles of section 2.4. Assume that the session $s \in \mathcal{S}$ consists of the embeddings $h_{ggc}^{\mathcal{G}^s} = [v_{s,0}, v_{s,1}, ...s_c] \in \mathbb{R}^{n \times d_1}$. The matrices $\mathcal{W}_4, \mathcal{W}_5, \mathcal{W}_6$ are used to adjust the weights of the corresponding vectors. Consequently, the layer is named $h_{se}^{\mathcal{G}^s} \leftarrow \textit{SessionAttentionNetwork}(h_{ggc}^{\mathcal{G}^s})$ throughout the rest of the thesis. This local embedding, s_c, encapsulates the user's immediate interests within the current session and corresponds to the last clicked item $h_{ggc}^{\mathcal{G}^s}\mathcal{T}$. The global embedding of a session is constructed by aggregating all node vectors within the session. Since different items within the session may have different levels of importance, a soft-attention mechanism is used to accurately represent the global session preferences. For each $h_{ggc}^{\mathcal{G}^s}$ item embedded in the session, a weight $v_{s,i}$ is calculated to indicate the importance of the item. This weight is then used to compute a weighted average of all item embeddings within the session, representing the global embedding s_g. The hybrid embedding is computed as $s_h = \text{MergeMLP}(s_g, h_{ggc_n}^{\mathcal{G}^s}, d^{\mathcal{I}'})$. Here, MergeMLP is represented as subsubsection 3.2.1. The combination of the attention layer, which is used to focus on the relevant parts of the input, and MergeMLP, which is used to process this information further, provides

an effective method to process the user session embedding ($h^{\mathcal{G}^s}_{ggc}$) [Wu+19]. Here $d^{\mathcal{I}'}$ is a hyperparameter referred to in this thesis as SESS-TERM-HIDDEN-SIZE. The resulting embedding vector for each user captures the distinctive features of their interactions with the items in their current session. This vector is then converted into a session embedding that represents the user's characteristics in relation to their interactions with the items. Then the user features are added $h^{\mathcal{G}^s} = \mathit{SessionUserCombination}(h^{\mathcal{G}^s}_{se}, \mathcal{M}^{\mathcal{U}})$, which is represented in Equation 3.11.

$$h^u_1 = (\mathcal{W}_7 h^{\mathcal{G}^s}_{se} + b_7)\mathit{Dropout}(0.2) \tag{3.11}$$

$$h^u_2 = (\mathcal{W}_8 u c^u_s + b_8)\mathit{Dropout}(0.6) \tag{3.12}$$

$$s_h = \mathcal{W}_9(\sigma(h^u_2 + h^u_1)) + b_9 \tag{3.13}$$

$$h^{\mathcal{G}^s} = \mathrm{ELU}(s_h h^{\mathcal{G}^s}_{se}) \tag{3.14}$$

The purpose of Equation 3.11 is to learn a function that updates the hidden vectors $h^{\mathcal{G}^s}_{se}$ with the user data $u \in \mathcal{M}^{\mathcal{U}}$ into an outgoing vector $h^{\mathcal{G}^s}$. In this combination, the embeddings of the user can be weighted using the CATEGORY-SHIFT c^u_s to ensure that the model assigns different weights to users who prefer to use the same categories. The CATEGORY-SHIFT allows capturing and including the influence of these user habits in the embeddings, where $c^u_s = \frac{c_s}{c_a/|c_a|}$. Here, c_s corresponds to CATEGORY-SHIFT per session and can be described as $c_s = \frac{|c_s`|}{|c_s|}$. Here, $|c_s|$ represents the length of the list of all categories per session, while $|c_s`|$ represents the length of the list of all categories per session when duplicate sessions cannot be counted. c_a represents CATEGORY-SHIFT during all sessions, and $|c_a|$ represents the length of that list. If the data set contains no categories, the weight of CATEGORY-SHIFT is set to 1 to exclude the influence of categories. This calculation can also be controlled by the hyperparameter USE-CATEGORY-SWITCH, which allows one to analyze the impact of categories on the modeling. If USE-CATEGORY-SWITCH is set to zero, then $c^u_s = 1$ also holds, which corresponds to a weighting of embeddings that is independent of categories. Accounting for the CATEGORY-SHIFT weights of different categories allows the model to capture individual user preferences and control their influence in the analysis. This approach allows for finer granularity in modeling and improves the accuracy and representation of user interactions. Here, $\mathcal{W}_7$, $\mathcal{W}_8$ and $\mathcal{W}_9$ correspond to the learnable weight matrices, and b_7, b_8 and b_9 correspond to the bias term. Furthermore, *Dropout* is a common technique used in NN training to avoid overfitting [Sri+14]. By eliminating random neurons in each layer, *Dropout* can help improve the

generalizability of a network [Sri+14]. This is a necessary step because the user data are already included in the hidden vector by TGR-STR and should therefore be considered less important. The function σ is a sigmoid function defined in Equation 2.20 that performs a nonlinear transformation on the input value. Furthermore, the function ELU is the exponential linear unit activation function defined in Equation 2.17.

The final embedding $h^{\mathcal{G}^s}$ for each user can be used to calculate the score. To make this embedding accessible for TGR-HIS layers, it is necessary to store it in $\mathcal{M}^{\mathcal{S}^u}$. The update of session memory is called *SaveSessionEmbeddings* in the algorithm. Here, $\mathcal{M}^{\mathcal{S}^u} = h^{\mathcal{G}^s} t^d$ is applied. Different weightings are applied to give less importance to older sessions. The process of combining the embedding vector $h^{\mathcal{G}^s}$ with the timestamp t^d is an important step in incorporating time information for the historical calculation of TGR-HIS. To compute the timestamp, t^d the timestamps of the interactions in the graph $\mathcal{G}^s$ are used. Note that these timestamps are in Unix format, representing the number of seconds since January 1, 1970 00:00:00 UTC. To properly weight the embedding $h^{\mathcal{G}^s}$, it is recommended to normalize the timestamp to the range [0,1]. For this purpose, the normalization transformation $t^d = \frac{t}{10^{10}}$ is used to reduce the magnitude of the timestamp t^d. This normalization exclusively serves to scale the time dimension and facilitates the weighting of embeddings based on their temporal relevance.

The combination of temporal information and embeddings opens up the possibility of incorporating past interactions into recommendation systems. This allows personalized and relevant recommendations to be generated based on the user's past behavior. To preserve this effect, this calculation can be limited to USE-SESSION-TIME-SAVE. If USE-SESSION-TIME-SAVE is set to zero, then $t^d = 1$ is applicable.

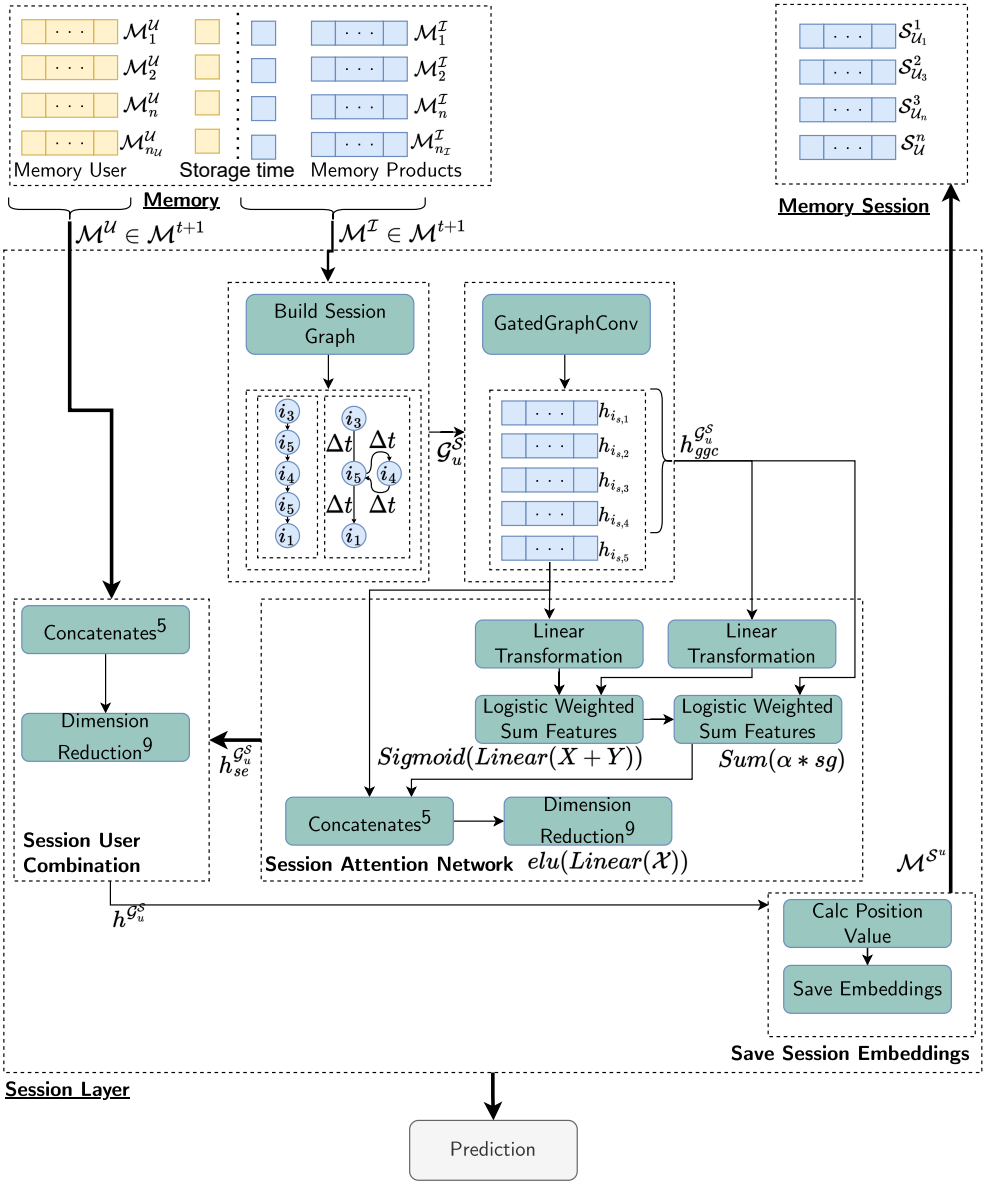


Figure 3.4.: The TGR Session Layer is a layer used in TGR to model a user's usage habits and create personalized recommendations. It uses session graphs to represent a user's interactions with items over a period of time. The algorithm creates an embedding vector for each user, representing the characteristics of their interactions with items, and stores it in memory for further steps in the recommendation system.

3.5. TGR Historical layer

This section gives a detailed exposition of the author's conception of the last layer Temporal Graph Recommendation Historical Layer (TGR-HIS). First, section subsection 3.5.1 explains the motivation behind the implementation of this layer. This explanation is followed in subsection 3.5.2 by describing the model implementation strategy. In addition, the formal definition of the model is presented in this section.

3.5.1. Foundation

TGR-HIS is the final hierarchical layer of the recommendation system TGR presented in this thesis. It uses the current session embedding $h^{\mathcal{GS}}$ and the historical embeddings $\mathcal{M}^{\mathcal{S}_h^u} \subset \mathcal{M}^{\mathcal{S}^u} \subset \mathcal{M}^{\mathcal{S}}$, which contain the historical sessions for user u. These embeddings have been created by TGR-SES.

The rationale for including historical session data is explained in section 2.3. Sequential recommendation systems generally utilize historical session data to offer personalized recommendations to individual users.

In general, this information is used to generate recommendations for the user based on their previous interactions with the system. The goal is to provide the user with a personalized and relevant experience by recommending interaction opportunities that match their interests and preferences [Cha+21; Wu+20a; Gao+21].

3.5.2. Implementation

This algorithm creates a relationship between the user's historical and current sessions. Here, $\mathcal{M}^{\mathcal{S}^u}$ represents the user's historical sessions, while $\mathcal{S}_c^u$ represents the current session. The current session $\mathcal{S}_c^u$ is not considered historic. Consequently, the historical session data is represented as $\mathcal{M}^{\mathcal{S}_h^u} = \mathcal{M}^{\mathcal{S}^u} \setminus \mathcal{S}_c^u$.

It is imperative to consider combining historical data with the current session if $|\mathcal{M}^{\mathcal{S}_h^u}| > 0$. If this condition is true, the hidden vector $h_{ha}^{\mathcal{H}^u}$ can be calculated by applying the function *MultiHeadAttention* with $\mathcal{Q} = \mathcal{S}_c^u$, $\mathcal{K} = \mathcal{V} = \mathcal{M}^{\mathcal{S}_h^u}$ as input parameters.

$$h_{ha}^{\mathcal{H}^u} = \mathit{MultiHeadAttention}(\mathcal{Q}, \mathcal{K}, \mathcal{V})$$

Without historical data, $h_{ha}^{\mathcal{H}^u}$ defaults to $\mathcal{S}_c^u$.

MultiHeadAttention is chosen because it allows better handling of historical sessions with the current session [Vas+17; Ros+20; Wu+19]. It can calculate different weights

Algorithm 4: TGR Historical Layer

Data: Storage $\mathcal{M}$, Session Storage Embedding ($\mathcal{M}^{\mathcal{S}^u}$

Result: Score per Item

Algorithm: TGR Historical Layer

while $u \in \mathcal{B}$ **do**

  $\mathcal{M}^{\mathcal{S}^u} \leftarrow \mathcal{M}^{\mathcal{S}^u} \in \mathcal{M}^{\mathcal{S}}$;

  $\mathcal{S}_c^u \leftarrow h^{\mathcal{G}^s}$;

  $\mathcal{M}^{\mathcal{S}_h^u} \leftarrow \mathcal{M}^{\mathcal{S}^u} \setminus \mathcal{S}_c^u$;

  if $|\mathcal{M}^{\mathcal{S}^u}| \geq 2$ **then**

    $h_{ha}^{\mathcal{H}^u} = \mathit{MultiHeadAttention}(\mathcal{S}_c^u, \mathcal{M}^{\mathcal{S}_h^u}, \mathcal{M}^{\mathcal{S}_h^u})$;

  else

    $h_{ha}^{\mathcal{H}^u} = \mathcal{M}^{\mathcal{S}^u}$;

  end

  $h_u^* \leftarrow \text{MergeMLP}(h_{ha}^{\mathcal{H}^u}, \mathcal{S}_c^u, d^{\mathcal{I}'})$;

end

of historical data, manage intricate relationships between data, and ensure that all relevant information from historical sessions is captured and incorporated into the recommendation. This results in better prediction of user preferences and improved recommendation accuracy [Vas+17; Ros+20; Wu+19].

A concatenation of $h_{ha}^{\mathcal{H}^u}$ and $\mathcal{S}_c^u$ is performed to create a weighted association with the current session, where the current session is weighted more prominently. This is done by computing the vector representation of $h_{ha}^{\mathcal{H}^u}$, where $h_{ha}^{\mathcal{H}^u} = \text{MergeMLP}(h_{ha}^{\mathcal{H}^u}, \mathcal{S}_c^u, d^{\mathcal{I}'})$. Here $d^{\mathcal{I}'}$ is the hyperparameter SESS-TERM-HIDDEN-SIZE.

3.6. TGR Score Function

This section presents the score function, which is an integral part of calculating the final score for each item for each user. It has a significant impact on the quality of recommendations. The score function processes an embedding vector for each user, yielding a score reflecting the item's relevance. This relevance is represented in the range $[0, 1]$, where these numbers indicate the probability. This score is used to generate a ranked list of recommended items to be displayed to the user.

Theoretically, such a score function can be constructed as a linear combination of the

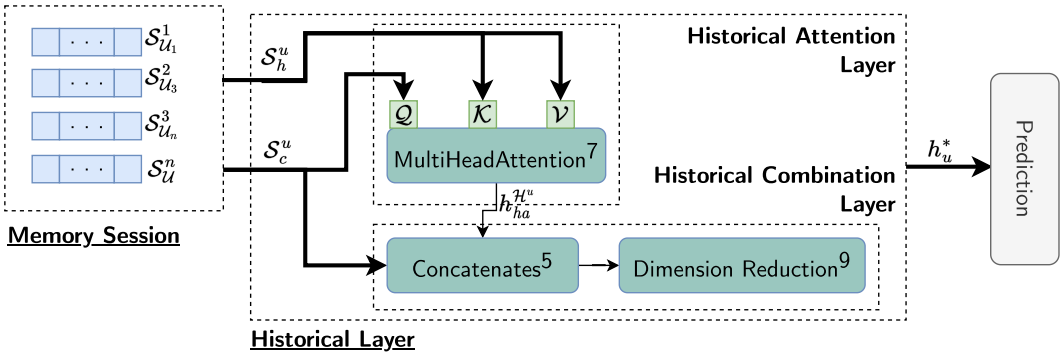


Figure 3.5.: The TGR Historical Layer combines the current session graphs of users with their historical session graphs if they are available. This algorithm uses *MultiHeadAttention* to compare the historical and current session graphs. The combined session graphs are then processed and multiplied by items to obtain a score for each item.

characteristics of the user and the item Pan et al. [Pan+20]. However, in this thesis, learnable parameters are used to ensure greater accuracy and customization [Wu+20a].

The current vector representation, $\mathcal{U}_{embedding}$, is combined with the elements, $\mathcal{M}^{\mathcal{I}'}$, to compute a score as described in Equation 3.15, Figure 3.6 and algorithm 5.

Algorithm 5: TGR Score Function

Data: $\mathcal{U}_{embedding}$

Result: Score per item (i)

$\alpha \leftarrow \text{Linear}(\sigma(\mathcal{U}_{embedding}))$;

$a \leftarrow \text{Linear}(sun(\alpha \times \mathcal{U}_{embedding}))$;

$\hat{z}\sigma(\text{MergeMLP}(a, \mathcal{M}^{\mathcal{I}'}, 1)$;

$\hat{y}_i \leftarrow \hat{z}_i$;

$\mathcal{U}_{embedding}$ could be the result of TGR-HIS, as the model can use different layers, similar to the hyperparameter RUN-MODE, so $\mathcal{U}_{embedding}$ corresponds to different embeddings.

If RUN-MODE is equal to STRUCTURAL, only the TGR-STR layer is used, and consequently $\mathcal{U}_{embedding} = h^{n\prime}_{\mathcal{V}}$. If RUN-MODE is equal to SESSION, the TGR-STR layer is used in conjunction with STRUCTURAL, so $\mathcal{U}_{embedding} = h^{\mathcal{G}^s}$. For RUN-MODE equal to HISTORY, all three layers are used, so $\mathcal{U}_{embedding} = h^*_u$.

$$\begin{aligned} \alpha &= \text{Linear}(\sigma(\mathcal{U}_{embedding}) \\ a &= \text{Linear}(\sum(\alpha \cdot \mathcal{U}_{embedding})) \\ \hat{z} &= \sigma(\text{MergeMLP}(a, \mathcal{M}^{\mathcal{I}'}, 1))\hat{y}_i \leftarrow \hat{z}_i \end{aligned} \tag{3.15}$$

In Equation 3.15, Linear represents a linear layer as defined in subsubsection 3.2.1, σ denotes the sigmoid activation function, which is detailed in Equation 2.20.

Here, $\mathcal{U}_{embedding}$ denotes the user embedding, and $\mathcal{M}^{\mathcal{I}'}$ is a matrix of item features. The operation $\cdot$ is an element-wise multiplication between vectors.

The user embedding $\mathcal{U}_{embedding}$ is processed with a linear layer and a sigmoid activation function. Then a weighted average a is calculated using α to form the embedding of the elements to be recommended. This result is then processed with a linear layer and finally scaled with a sigmoid activation function to compute a weight for the elements in the memory matrix $\mathcal{M}^{\mathcal{I}'}$. This combination of layers allows the network to model user-item relationships while reducing the complexity of user embedding and feature relevance scoring.

The sigmoid function was chosen as the evaluation function because of its particular suitability for recommendation systems. It provides a straightforward way to convert an evaluation result into a relative score within the interval $[0, 1]$, thus generating personalized recommendations [DSC22].

In particular, the sigmoid function provides a continuous transition curve that maps complex calculations to a simple, interpretable score. This is critical as modern recommendation systems leverage rich data sources and require meaningful information to generate personalized recommendations [DSC22].

By integrating these elements into the score function, the model can consider user preferences and generate recommendations for items most likely to meet their needs.

3.7. Training of TGR

This section outlines the training process for TGR. This section does not include the description of TGR itself. The definition given in the previous sections applies.

The training process is summarized in Figure 3.7. The data set $\mathcal{X}$ is arranged in ascending chronological order, allowing the model to first learn from the earliest interactions. In the TGR-STR layer, the batch data $\mathcal{B}$ is used to update the memory module, updating the embeddings for each node in the dynamic graph (1-3). The

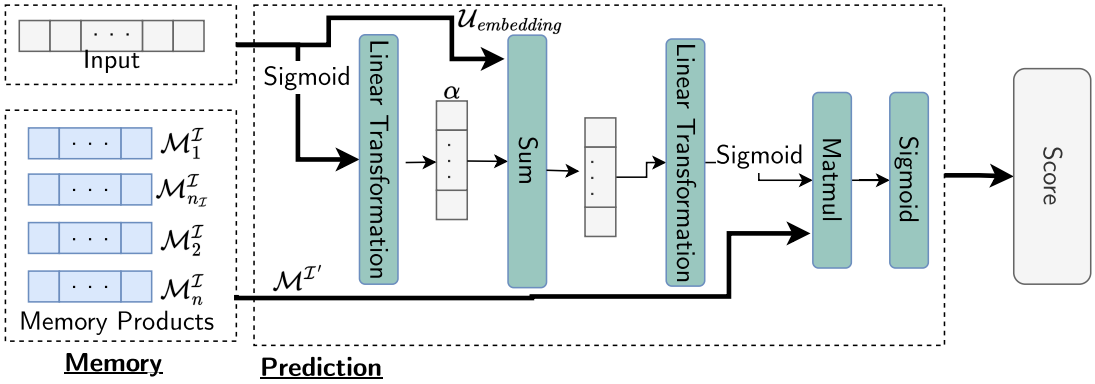


Figure 3.6.: The algorithm TGR Historical Layer combines current session graphs of users with historical session graphs, if available. This algorithm uses *MultiHeadAttention* to compare the historical and current session graphs. The combined session graphs are then processed and multiplied by items to obtain a score per item.

complexity of the training strategy escalates because the memories $(\mathcal{M}, \mathcal{M}^{\mathcal{S}})$ do not directly affect the loss, so they do not receive a gradient. This requires updating the memory before updating the batch interaction prediction, a strategy already demonstrated by Rossi et al. [Ros+20] (TGN). The updated memory $\mathcal{M}^{t+1}$ is used in the TGR-SES layer to construct a session graph and compute a user-specific embedding $(h^{\mathcal{G}^s})$, which is then stored in session memory $\mathcal{M}^{\mathcal{S}}$ (4-6). This embedding is merged with previous embeddings $h^{\mathcal{G}^s}$ of the same user to derive an embedding that spans all sessions. This computation is performed in the last layer, TGR-HIS (7-9). Arrows 10 and 11 represent the hyperparameter RUN-MODE, which activates different layers depending on its setting. It should be emphasized that all computations in a particular batch $\mathcal{B}$ access the same memory state. Although the memory $(\mathcal{M}, \mathcal{M}^{\mathcal{S}})$ is up-to-date from the perspective of the first interaction in the batch, as it contains information on all previous interactions in the graph, it is considered obsolete from the perspective of the last interaction in the batch since it does not contain information on previous interactions in the same batch [Ros+20]. A large BATCH-SIZE should be avoided. Otherwise, all predictions will be made based on the initial zero memory, especially if the batch size is equal to the size of the data set. Previous research has shown that a BATCH-SIZE of 200 is a good compromise between the speed and granularity of updating [Ros+20]. This limitation affects each module (TGR-STR,TGR-SES,TGR-HIS) individually. Of course, the building module has access

to the updated memory of the previous module.

Following the calculation of the score, the Binary Cross Entropy Loss (BCE Loss) function is used to optimize the gradients of the model. This function quantifies the average negative logarithm of the agreement of the weighted prediction target for all instances [Li+23]. The mathematical formulation of the BCE Loss function is defined in Equation 3.16.

$$\mathcal{L} = BCE(y, \hat{y}) \tag{3.16}$$

$$= -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)) \tag{3.17}$$

In BCE Loss y is the target which can be $y = 0 \vee y = 1$ and $\hat{y}$ is the calculated score of an item which can be $\hat{y} \in [0, 1] \subset \mathbb{R}$ can be, and log is the natural logarithm.

In recommendation systems, a loss function is used to evaluate the model predictions against its binary targets [Gao+21; Li+23]. In these systems, a user's interaction with an item can be represented as $\hat{y} = 1$, while a non-interaction can be represented as $\hat{y} = 0$. A non-interaction occurs with negative sampling and should be considered accordingly. Because BCE Loss calculates the average in all instances, it ensures that misclassifications, such as incorrect interaction predictions, are correctly captured [Li+23]. It is important to note that the BCE Loss function is often used in the context of positive and negative samples [Ros+20; Li+23].

This thesis will apply the loss function defined in Equation 3.18.

$$\hat{z}_{\mathcal{I}}^{p}, \hat{z}_{\mathcal{I}}^{n} = TGR(\mathcal{X}) \tag{3.18}$$

$$\hat{z}^{p} = \hat{z}_{\mathcal{I}}^{p} \cap \mathcal{I}^{p} \in \mathcal{B} \tag{3.19}$$

$$\hat{z}^{n} = \hat{z}_{\mathcal{I}}^{n} \cap \mathcal{I}^{n} \in \mathcal{B} \tag{3.20}$$

$$\mathcal{L} = \mathcal{L}^{p}(1, \hat{z}^{p}) + \mathcal{L}^{n}(0, \hat{z}^{n}) \tag{3.21}$$

In this case, $\hat{z}^{p}$ represents the positive scores of the positively selected items $\mathcal{I}^{p}$, and $\hat{z}^{n}$ represents the negative samples $\mathcal{I}^{n}$ [Ros+20]. This function aims to penalize a negative sample if it is predicted to be an interaction. Thus, the total loss function $\mathcal{L}$ is the sum of the loss functions for positive samples $\mathcal{L}^{p}$ and negative samples $\mathcal{L}^{n}$ [Ros+20]. Negative examples are not used to update the memory.

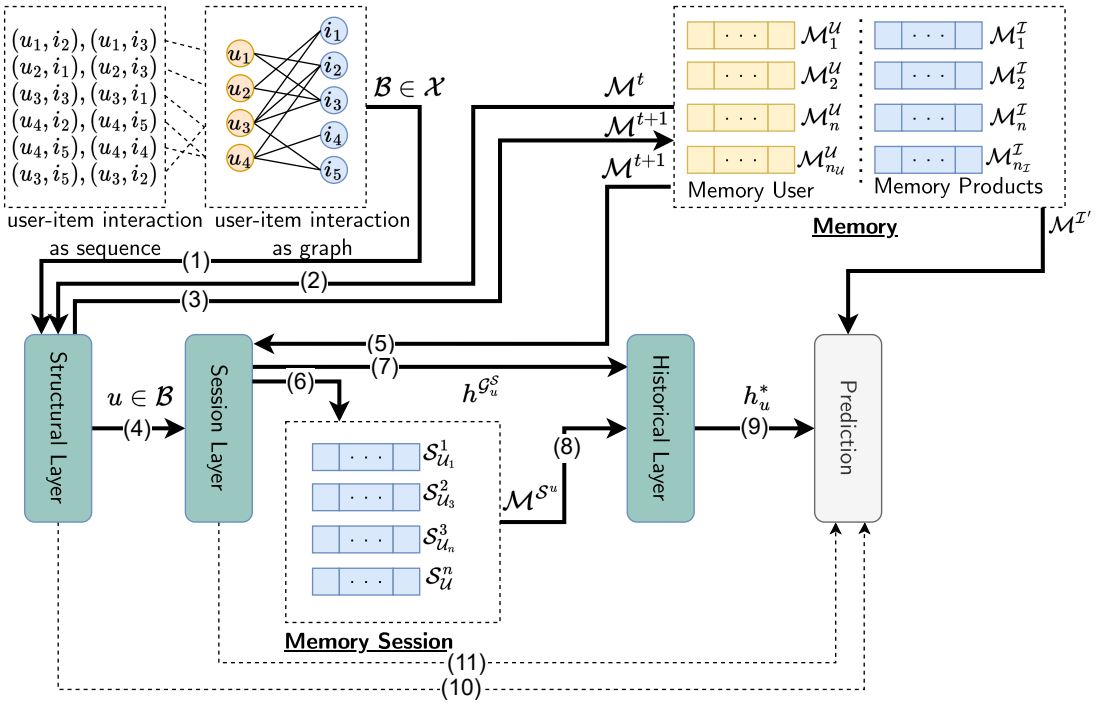


Figure 3.7.: Training process of the TGR model. Here it is shown that the data is used to update the memory module and the embeddings in the structural layer. A session graph is created in the session layer, and embeddings for each user are stored in the session memory. The history layer combines the embeddings across all sessions to obtain an overall embedding.

3.7.1. TGR Hyperparamter

This section presents the hyperparameters that significantly impact the performance and accuracy of the trained model. It is essential to carefully select these hyperparameters to achieve an optimal result [Gao+21]. Note that the dimensions determined by the data set used will be defined later for each specific dataset, and only a placeholder value is entered here.

The Table 3.1 presents an overview of the hyperparameters and their default values for the Temporal Graph Recommendation (TGR). It also includes the notation assigned to each hyperparameter, if applicable. In Table 3.1 STR, SES, and HIS abbreviate STRUCTURAL, SESSION, and HISTORY, respectively. Similarly, MS, LS, and R represent MOST-SIMILAR, LESS-SIMILAR, and MOST-SIMILAR. Furthermore, **Final** indicates whether the hyperparameter will be further analyzed in this thesis. If the column contains *no*, it

Hyperparameters	Default	Value set	Notation	Final
BATCH-SIZE	200	$\mathbb{N}$	$\lvert\mathcal{B}\rvert$	yes
DROP-OUT	0.1	$\mathbb{R}$		yes
LEARNING-RATE	$3e-4$	$\mathbb{R}$		yes
NEGATIVE-SAMPLER	MS	$\{\text{MS}, \text{LS}, \text{R}\}$		no
RUN-MODE	HIS	$\{\text{STR}, \text{SES}, \text{HIS}\}$		no
BACKPROP-EVERY-N	1	$\mathbb{N}$		yes
DATA-SPLIT-RATIO	0.6	$\mathbb{R}$		no
DATA-TEST-SPLIT-RATIO	0.2	$\mathbb{R}$		no
MIN-COUNT-USER	5	$\mathbb{N}$		no
MIN-COUNT-ITEM	5	$\mathbb{N}$		no
REMOVE-COMMON-USERS	0.05	$\mathbb{R}$		no
REMOVE-COMMON-ITEMS	0.05	$\mathbb{R}$		no
USE-EMBEDDING-USER	0	$\{0, 1\}$		no
USE-EMBEDDING-PRODUCT	0	$\{0, 1\}$		no
EMBEDDING-DIMENSION-USER	181	$\mathbb{N}$		no
EMBEDDING-DIMENSION-PRODUCT	412	$\mathbb{N}$		no
MEMORY-DIMENSION-USER	181	$\mathbb{N}$	$d^{\mathcal{U}}$	no
MEMORY-DIMENSION-PRODUCT	412	$\mathbb{N}$	$d^{\mathcal{I}}$	no
NUM-NEIGHBORS	10	$\mathbb{N}$	k	yes
GAT-HIDDEN	412	$\mathbb{N}$		yes
SESS-TERM-HIDDEN-SIZE	200	$\mathbb{N}$	$d^{\mathcal{I}'}$	yes
N-LAYER	1	$\mathbb{N}$		yes
N-HEAD	1	$\mathbb{N}$		yes
USE-CATEGORY-SWITCH	0	$\{0, 1\}$		no
USE-SESSION-TIME-SAVE	0	$\{0, 1\}$		no

Table 3.1.: Overview of hyperparameters and their default values for TGR

is intended for future research. Each hyperparameter will be explained in more detail below.

BATCH-SIZE controls the number of elements within a batch ($|\mathcal{B}|$), the standard value for this is 200. DROP-OUT controls the *Dropout* layers that are not explicitly specified. *Dropout* is a regularization technique used to prevent overfitting of the training data set [Sri+14]. The connections between neurons are chosen randomly and their weights are set to zero during training. A default value of $0.1 \in \mathbb{R}$ is used in TGR and is not discussed further.

The parameter LEARNING-RATE dictates the magnitude of weight adjustment during the training process section 3.7. This thesis uses a learning rate of $3e-4 \in \mathbb{R}$. The decision to use this constant is influenced by previous work [Ros+20]. Although it remains uncertain whether this is the best approach for the current model, it should be noted that no simulations were performed in this thesis. NEIGHBOR-SAMPLER is a hyperparameter that specifies how the model selects negative examples for training. The default is RANDOM. This is described in subsection 3.3.2.

RUN-MODE is a hyperparameter that specifies which layers to run in the model. A distinction is made between STRUCTURAL, SESSION, and HISTORY. The default value for RUN-MODE is HISTORY. This is described in detail in section 3.7.

The DATA-SPLIT-RATIO specifies the portion of the data set that will be used for training. This implementation uses a default value of $0.6 \in \mathbb{R}$. In addition, DATA-TEST-SPLIT-RATIO controls the portion of the data set used for the testing. This implementation uses the default value of $0.2 \in \mathbb{R}$. The remaining data are used for validation, which would also be $0.2 \in \mathbb{R}$. There is no check to determine whether a different split would produce superior results.

In addition to the split, MIN-COUNT-USER and MIN-COUNT-ITEM control the minimum number of times a user or an item must be present in the data set to be included in the model. This thesis uses a default value of $5 \in \mathbb{N}$ for both parameters. REMOVE-COMMON-USERS and REMOVE-COMMON-ITEMS control the minimum number of times a user or item must be present in the training data set to be included in model training. If the frequency of a user or item exceeds the upper threshold, the user or item is removed from the training data set. This process helps to cleanse the model by removing data points that would excessively influence the training. This allows the model to focus on more relevant patterns.

USE-EMBEDDING-USER and USE-EMBEDDING-PRODUCT control the utilization of user and item embeddings. These embeddings are used when a record lacks user or item

information. Specific details regarding the utilization of these parameters are explained in the relevant sections that describe the experiments conducted in the thesis. The objective is to assess whether the embeddings exhibit behavior similar to real data. If this parameter is set to 1, the model will use embeddings for users or items. EMBEDDING-DIMENSION-USER and EMBEDDING-DIMENSION-PRODUCT control the size of these embeddings.
It is recommended that EMBEDDING-DIMENSION-USER = MEMORY-DIMENSION-USER = $d^{\mathcal{U}}$, EMBEDDING-DIMENSION-PRODUCT = MEMORY-DIMENSION-PRODUCT = $d^{\mathcal{I}}$. This option is not pursued further since another layer must be added to accommodate different dimensions. MEMORY-DIMENSION-PRODUCT and MEMORY-DIMENSION-USER control the size of the vectors for users and items, as well as the size of the corresponding memory. NUM-NEIGHBORS controls the number of neighbors to consider per node, as described in subsection 3.3.2. Indeed, while increasing the number of neighbors can enhance the accuracy of the prediction, it also increases the computational expense. In the thesis, NUM-NEIGHBORS = $k = 10$ is used, as this value has been determined to be suitable in previous research [Ros+20]. Whether this is also the case in this thesis has not been further investigated.

GAT-HIDDEN controls the size of the hidden layer in GGS-NN used in TGR-SES. A larger hidden layer can capture complex interactions between items in the session graph, but it also increases the model's size and the training overhead. This is described in more detail in subsection 3.4.2. This thesis uses GAT-HIDDEN = MEMORY-DIMENSION-PRODUCT because otherwise a linear layer would have to be added to the session graph, resulting in a different dimension. More research is needed to determine whether this is a positive effect.

SESS-TERM-HIDDEN-SIZE controls the size of the dimension reduction in TGR-SES, which is described in detail in subsection 3.4.2. The USE-CATEGORY-SWITCH hyperparameter controls whether categories should be used when combining session graph embeddings with user embeddings. A detailed description can be found in subsection 3.4.2. Furthermore, USE-SESSION-TIME-SAVE controls whether the variable t^d should be considered when saving the session. More information can be found in subsection 3.4.2.

4. Datasets

Data sets are an essential foundation for the development, training, and evaluation of recommendation systems. They identify user preferences and facilitate prediction. Without adequate and reliable data, recommendation systems cannot be trained effectively, resulting in inaccurate recommendations. Therefore, it is imperative to collect and process high-quality data sets [Wu+20a; Kaz+19; Zhu+22]. This chapter introduces the data sets used to train TGR described in chapter 3.

The data sets used in this thesis contain extensive information about users, items, and interactions between them. These interactions can be derived from item purchases or user interactions. By leveraging these interactions, it becomes possible to construct a graph that effectively represents the intricate relationships between users and items. A comprehensive description of this process is provided in subsection 2.5.1.

A common challenge in processing these data sets is the existence of incomplete and inconsistent data. As a consequence, it becomes imperative to perform data cleaning and preprocessing procedures to adequately prepare the data for the training of the recommendation system [Wu+20a; Gao+21]. These procedures can include the elimination of duplicate entries, the input of missing values, or normalization of the data [Wu+20a; Gao+21].

This chapter covers the preprocessing steps of four data sets: *YooChoose*, *Diginetica*, *RetailRocket*, and *Refurbed*. The preprocessing techniques applied to each are referenced in section 4.3, section 4.2, section 4.4, and section 4.5, respectively.

Section 4.6 provides a comprehensive synthesis and comparison of the data sets discussed above. This section provides an overview of the benefits of each data set and how they specifically contribute to the objectives and findings of this thesis.

4.1. Requirements

This section provides an overview of the preprocessed data set $\mathcal{X}$, presented in an appropriate format. The variable $\mathcal{X}$ corresponds to one of the previously mentioned

data sets and is processed in the corresponding section. In preparation for the following sections dealing with preprocessing, it is essential to clarify the required attributes of the $\mathcal{X}$ data set in order to ensure the correct functioning of the proposed algorithm. It applies $(\mathcal{U}, \mathcal{I}, \mathcal{E}) \in \mathcal{X}$ for all data sets. The users, represented by $\mathcal{U}$, are the individuals who use the recommendation system and their details are detailed in subsection 4.1.2. The items described in subsection 4.1.3 are the items recommended by the system. The edges, denoted as $\mathcal{E}$, symbolize the interactions between users and items and are discussed in subsection 4.1.1.

4.1.1. Edges

The edge properties described in Equation 4.1 are an essential requirement for the present model. Additionally, for each edge $e \in \mathcal{E} \in \mathbb{R}^{d^{\mathcal{E}}}$, certain properties $\mathcal{E}'$ must exist to allow unique interactions between the user and the element. These properties include the *user ID* u_{id}, the *item ID* i_{id}, the interaction time $t^{(u,i)} \in \mathcal{E}'$, a unique ID e_{id} for the edge and an ID s_{id} associated with a session. If there are no session data in the data set, *session ID* is equivalent to the corresponding u_{id} ($\forall s_{id} \in \mathcal{S}, s_{id} = u_{id}$). It holds that $\mathcal{E}' \subseteq \mathcal{E}$.

To guarantee suitability for machine learning, it is necessary to transform the potentially irregular u_{id} into a numeric representation ranging from 0 to the total number of distinct users, denoted by $|\mathcal{U}|$. This transformation makes *user IDs* numeric, making them more compatible with machine learning algorithms. It is important to note that the IDs reflect the IDs of the corresponding matrix, not the IDs of the data publishers. For instance, the user $u_3 = \mathcal{U}_{[u_{id},]}$, if $u_{id} = 3$. The same transformation considerations apply to i_{id} as to u_{id}. Consequently, i_{id} is also transformed to range from 0 to the length of the different items.

The s_{id} indicates the interactions that occur within a session. Therefore, s_{id} is transformed on a per-u_{id} basis to range from 0 to $|s_{id u_{id}}|$, starting at 0 for each user. This transformation simplifies the access within the model.

$$\{\mathcal{E} \in \mathbb{R}^{d^{\mathcal{E}}} \mid u_{id} \in \mathbb{N}, i_{id} \in \mathbb{N}, t^{(u,i)} \in \mathbb{R}, s_{id} \in \mathbb{N}, e_{id} \in \mathbb{N}, \ldots\} \in \mathcal{E} \tag{4.1}$$

4.1.2. User

Including characteristics in this case is optional, however, it is expected that $\mathcal{U} \in \mathbb{R}^{d^{\mathcal{U}}}$. In the absence of user data in the dataset, an embedding layer is used. An embedding layer can be conceptualized as a look-up table that maps discrete inputs to continuous vectors

[Mik+13]. Formally, the embedding of an input u with index u_{id} can be represented as $u_{u_{id}} \in \mathbb{R}^{d^{\mathcal{U}}}$, where $|\mathcal{U}|$ denotes the embedding size. The weights of the embedding layer can be represented as a matrix $\mathcal{W} \in \mathbb{R}^{(\mathcal{U}_c \times |\mathcal{U}|)}$, where $\mathcal{U}_c$ is the number of users. Thus, the embedding of an input u with index u_{id} can be computed as $u_{u_{id}} = \mathcal{W}_{u_{id}}$ [Mik+13; BS18].

During training, the weights in $\mathcal{W}$ are optimized to ensure an effective translation of the input values into continuous vectors. A loss function is used to measure the deviation between the predicted embedding and the actual embedding [Mik+13; BS18]. In this context, the $\mathcal{U}$ data change from epoch to epoch. This simulates the evolution of the dynamic user graph. It follows that $u^t_{u_{id}} \neq u^{t+1}_{u_{id}}$.

4.1.3. Items

Any considerations the user makes about the items are representative. However, it is expected that $\mathcal{I} \in \mathbb{R}^{d^{\mathcal{I}}}$.

If the data set does not provide explicit information about the items, an embedding layer can be used to represent them. Similar to the embedding layer for users, the embedding layer for items can be conceptualized as a lookup table that transforms discrete inputs into continuous vectors.

4.2. Diginetica

Diginetica is a publicly available data set that provides valuable information on customer interactions with an e-commerce store [DIG16]. These interactions include both purchases made and interactions with individual items. It should be noted that this data set does not contain sensitive customer information, such as personally identifiable information or financial information. However, it contains information such as the date and time of each interaction and the type of interaction [DIG16]. These features make the data set particularly attractive for the research of recommendation systems, as it provides a rich collection of interaction data. Therefore, it is ideal to simulate recommendation systems in a realistic environment [DIG16; Wu+20a].

The following section describes this data set in detail and explains the processing methods used in this thesis.

The entire section is based on data from *Diginetica* [DIG16], so the source is not repeated throughout.

4.2.1. Distribution of data

This section describes the *Diginetica* data set in detail. The interpretation and preparation of the data set is central to the evaluation and discussion of the results, and therefore critical to the subsequent experiments with this data set. To provide a more detailed understanding of the data, this section describes the structure of the data set. First, the interaction data will be described, hereafter referred to as *Edges*, while the item data will be referred to as *Items*. As described above, the data set does not contain any user data, so *Users* will not be explained.

Edges

This data set, which describes interactions on the website, consists of several files, each with different attributes and interaction types. In the context of this thesis, the focus was specifically on the *train-item-views* data set. This particular data set is chosen for its large amount of interaction information, which is crucial for the development of TGR. It provides a substantial collection of user-item interactions, allowing for effective modeling of the sequence and patterns of these interactions. Although other data sets that represent clicks, purchases, or queries have their own value, analyzing them would require a more extensive study. Such an analysis is beyond the scope of this thesis, especially considering that TGR is not designed to accommodate different types of interactions.

The attributes of the *train-item-views* data set are as follows.

userId (int64) This column contains a unique ID for each user who has interacted with the site. This ID is called u_{id} in this thesis.

sessionId (int64) This column contains a unique ID for each session on the site, hereafter referred to as s_{id}.

itemid (int64) This column contains a unique ID for each item on the site, which will be referred to here as i_{id}.

timeframe (int64) This column contains the time in milliseconds since the first request in a session. Considering the timeframe of user interactions allows modeling sequential patterns and predicting future interactions.

eventdate (string) This column contains the session date in the format *y-m-d*. Including the session date allows further contextualization of user interactions and can be used to model seasonal effects or time-dependent trends, for example.

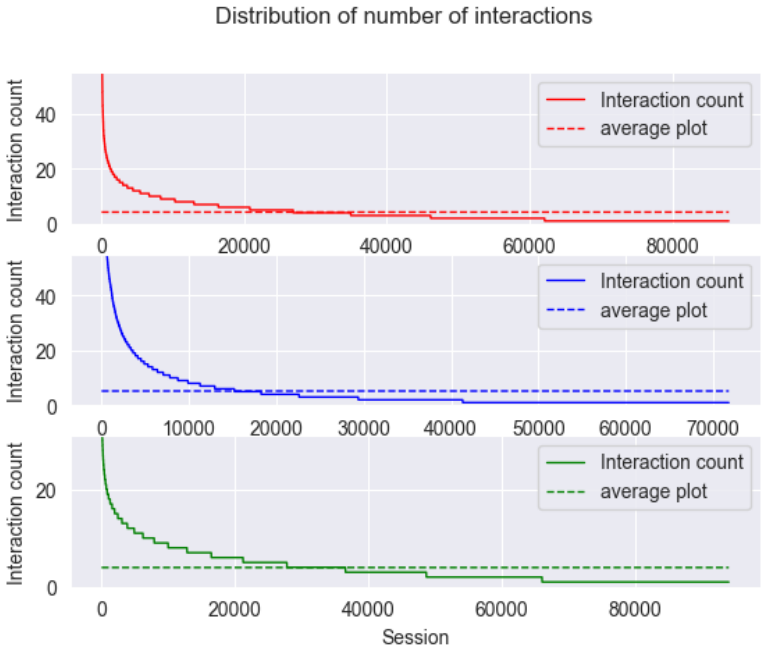


Figure 4.1.: Distribution of the data set *Diginetica*

The Figure 4.1 shows the distribution of the interactions of users (u_{id}), items (i_{id}), and sessions (s_{id}). To keep the graph readable, the most frequent 0.01% are not shown. All statistical information is listed in Table 4.1.

	$\lvert u_{id}\rvert$	$\lvert i_{id}\rvert$	$\lvert s_{id}\rvert$
count	87934	71799	94106
mean	4.24	5.19	3.96
min	1	1	1
99%	23	51	19
99.5%	29	71	23
max	238	531	81
Δ99%	791	703	833
Δ99.5%	408	357	393
Δmin5%	61074	53543	66307

Table 4.1.: Distribution table of the data set *Diginetica*

Table 4.1 shows the frequency distribution of $(u_{id}, i_{id}, s_{id}) \in \mathcal{E}$ in the data set *Diginetica*.

The variable $|u_{id}|$ indicates the frequency per unique *user ID* in the data set, $|i_{id}|$ indicates the frequency of unique *item IDs* and $|s_{id}|$ indicates the frequency of unique *session IDs*. For each variable, the quantile values of the minimum, maximum, mean, and 99% and 99.5% are provided. The data set contains a total of 87934 *user IDs*, 71799 *item IDs*, and 94106 *session IDs*. The average user purchased 4.24 items, while the average session contains 3.96 items.

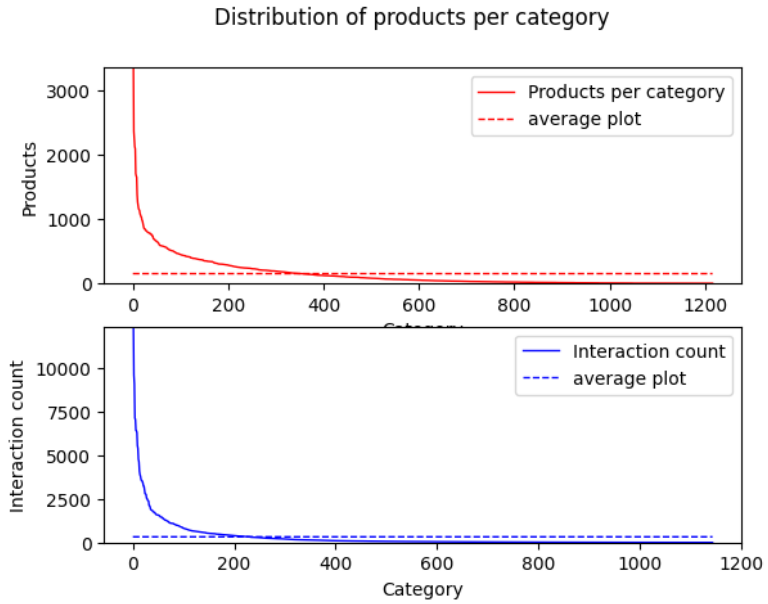


Figure 4.2.: Distribution of categories per item and interactions per category of the data set *Diginetica*

The quantile 99% indicates that 99% of the users purchased a maximum of 23 items, while 99% of the sessions contain a maximum of 19 items. The 99.5% quantile indicates that 99.5% of users interact with a maximum of 29 items, while 99.5% of sessions contain a maximum of 23 items. The maximum values for users, items, and sessions are 238, 531, and 81, respectively. The delta values show the differences between the values of the quantiles 99% and 99.5% and the maximum. These differences are important in identifying outliers in the data set, which can then be filtered and corrected with the appropriate hyperparameters. The delta values for the 99.5% quantile are 791, 703, and 833, while the delta values for the 99.5% quantile are 408, 357, and 393. Finally, the minimum delta 5% indicates how many IDs are in the lower part of the data set. In this case, 61074 *user IDs*, 53543 *item IDs*, and 66307 *session IDs* are in the lower 5% of the

data set. The quantiles shown here indicate the influence of the hyperparameters MIN-COUNT-USER, MIN-COUNT-ITEM = 5, REMOVE-COMMON-USERS, REMOVE-COMMON-ITEMS = 0.1 | 0.05.

The Figure 4.2 shows the distribution of categories per item and the distribution of interactions per category. In this table, it is noticeable that many items belong to the same category. It can be concluded that in this data set, user switching to items of other categories is less frequent. Therefore, the influence of category switching may be weaker in this data set. In Figure 4.3 the correlation matrix is displayed graphically. Analysis of the correlation matrix shows that there is no statistically significant correlation between the interactions. What is relevant in this context is the correlation between time and i_{id}, as this allows conclusions to be drawn about the customer's buying behavior. The other variables are less relevant because a correlation between s_{id} and u_{id} is obvious and a correlation between identifiers is not significant.

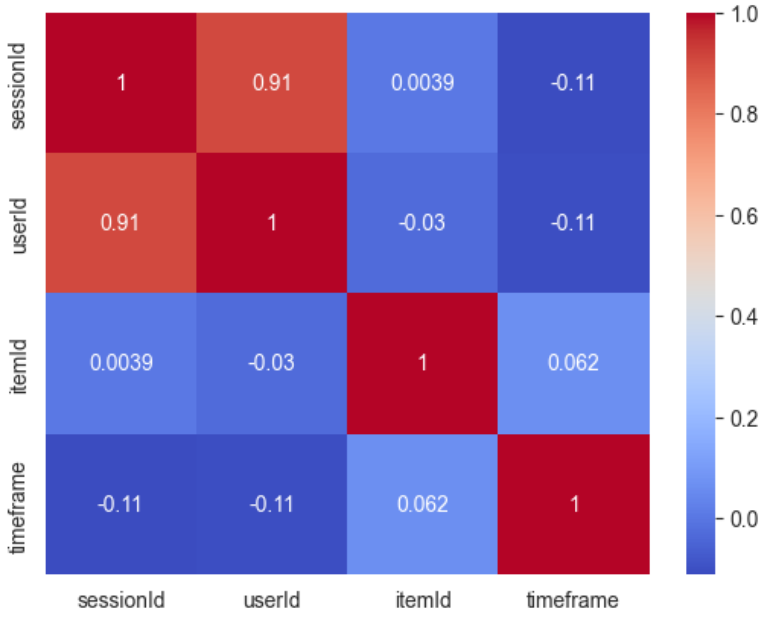


Figure 4.3.: Correlation matrix based on edge data of the data set *Diginetica*

Items

The item data in this data set is simple and is presented in the following list.

itemid (int64) a unique ID for each item on the web site, this is also referred to i_{id} in

this thesis.

pricelog2 (int64) log-transformed item price

product.name.tokens (string) comma separated hashed item name tokens

4.2.2. Data preprocessing

This section details the preprocessing of the *Diginetica* data set. Pre-processing can have a significant impact on the performance of the model [FC20]. It is important that the data preprocessing is done similarly as for the other data sets to achieve the best possible comparability. The influence of different pre-processing on the model TGR will not be further investigated.

Edges

userId (int64) The *user ID* is transformed from 0 to the length of the different users, accordingly $u_{id} \in [0, |\mathcal{U}|]$.

itemid (int64) The *item ID* is transformed from 0 to the length of the different items, accordingly $i_{id} \in [0, |\mathcal{I}|]$.

session (int64) The *session ID* is transformed from 0 to the length of the different session per user $s_{id} \in [0, |s_{id_{u_{id}}}|]$

timestamp (int64) *eventdate* to unix timestamp + *timeframe*, *eventdate* will not be processed any further and will be removed from the data set.

For u_{id}, i_{id}, and s_{id}, the considerations and transformations described in subsection 4.1.1 apply. Time information consists of two columns: *eventdate* and *timeframe*. *eventdate* indicates the date of the session, while *timeframe* indicates the time since the first query in a session. To make this time information more useful for machine learning, it is converted to Unix timestamps. The timestamp of *timeframe* is resolved in milliseconds and added to the Unix timestamp of *eventdate*. Converting all temporal data into Unix timestamps is a necessary step. This process ensures a consistent format that makes it possible for the model to infer time-based properties, such as the interval between two interactions. Importantly, this format remains independent of the original time zones or formats contained in the raw data. The Unix timestamp proves to be particularly suitable for this purpose because of its numerical and continuous nature. This allows efficient storage, sorting, and comparative analysis of time information.

Items

During data processing, item names are first considered as a list of comma separated hash tokens and stored under the attribute *product.name.tokens*. To use this information more effectively, each hashtoken is now treated as a separate attribute. Each new attribute is prefixed with *product.name* and named *product.name.list*. In this way, a total of 25 new attributes are created for each item name, ranging from *product.name.0* to *product.name.24*. For item names shorter than 25 tokens, the missing attributes are filled with zeros. The item names are defined as 64-bit integer types. Since only the last 10 tokens of item names are relevant in this data set and there are a total of 895 items in the data set, the attributes *product.name.10* through *product.name.24* and the attribute *product.name.tokens* are removed from the data set. To categorize the items, the i_{id} is transformed into $i_{id} \in [0, |\mathcal{I}|]$, the same as $i_{id} \in \mathcal{E}$.

4.3. YooChoose

The data set *YooChoose* is a publicly accessible data set that includes data on user clicks on an e-commerce website [Yoo22]. It will be focused on *YooChoose 1/4*. It provides information on a range of user activities, including viewing items, adding items to shopping carts, and purchasing items [Yoo22]. It is important to note that the *YooChoose* data set has been anonymized to ensure that it does not contain personally identifiable information about users or items with which they interact, which implies that no user and item pre-processing and analysis can be conducted [Yoo22].

The entire section is based on data from *YooChoose* [Yoo22], so the source is not repeated throughout.

4.3.1. Distribution of data

This subsubsection describes the *YooChoose* data set in detail. The data set is central to the analysis and interpretation of the results and therefore plays a crucial role in this paper. To gain a deeper understanding of the data, this subsubsection presents the structure of the data set in detail. This is crucial, as it allows us to examine how the model performs with the minimal requirements.

Edges

The data set describing the edges has the following attributes.

sessionId (int64) This column contains a unique ID for each session on the site, hereafter referred to as s_{id}.

itemid (int64) This column contains a unique ID for each item on the site, which here will be referred to as i_{id}.

time (int64) indicates the date and time when the event took place.

The Figure 4.4 shows the distribution of the interactions of users (u_{id}), items (i_{id}), and sessions (s_{id}). To keep the graph readable, the most frequent 0.01% are not shown. All statistical information is listed in Table 4.2.

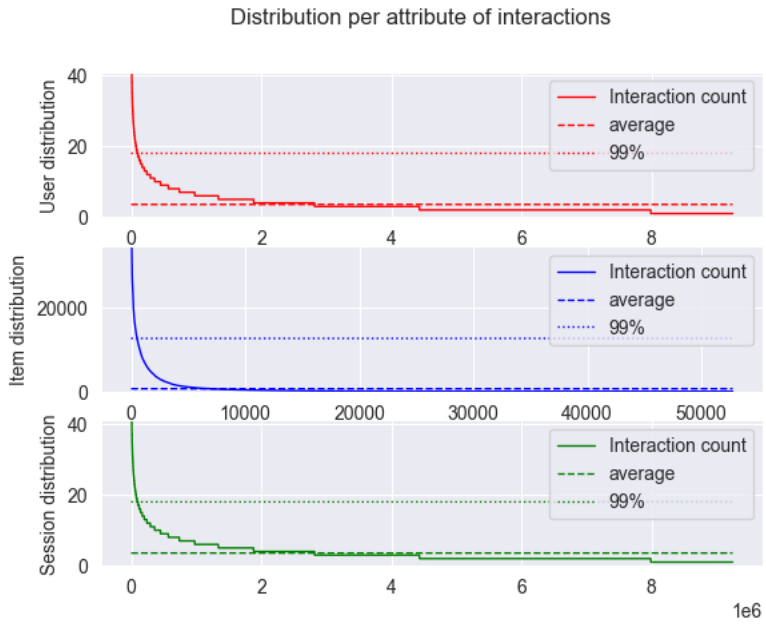


Figure 4.4.: Distribution of the data set *YooChoose*

The frequency distribution of the variables u_{id}, i_{id} and s_{id} in the data set *YooChoose* is presented in Table 4.2. The variable $|u_{id}|$ represents the frequency per unique *user ID*, $|i_{id}|$ represents the frequency of unique *item IDs*, and $|s_{id}|$ represents the frequency of unique *session IDs*. The table includes the values of the minimum, maximum, mean, 99% percentile, and 99.5% percentile for each variable.

The data set contains a total of 9249729 *user IDs*, 52739 *item IDs*, and 9249729 *session IDs*. It is important to recognize that the data set contains only *session IDs*, so $u_{id} = s_{id}$ applies to the entire table. Therefore, this section focuses primarily on u_{id}, but also applies to s_{id}.

	$\lvert u_{id}\rvert$	$\lvert i_{id}\rvert$	$\lvert s_{id}\rvert$
count	9249729	52739	9249729
mean	3.57	625.8	3.57
min	1	1	1
99%	18	12621.46	18
99.5%	24	17634.60	24
max	200	147419	200
Δ99%	90638	528	90638
Δ99.5%	42142	264	42142
Δmin5%	7367045	14775	7367045

Table 4.2.: Distribution of the data set *YooChoose*

On average, each user in the data set purchased 3.57 items. The 99% percentile value indicates that 99% of the users bought a maximum of 18 items, while the 99.5% percentile value indicates that 99.5% of the users interacted with a maximum of 24 items. The maximum values for users and items are 200 and 147419, respectively.

The data show that some items are disproportionately present in many sessions. Delta values highlight the differences between the 99% and 99.5% percentile values and the maximum values. These differences are critical for identifying outliers in the data set, which can then be filtered and corrected using appropriate hyperparameters.

For the 99% percentile, the delta values are 90638 for users and 528 for items, while for the 99.5% percentile, the delta values are 42142 for users and 264 for items. This information helps to understand the magnitude of the outliers in the data set.

Finally, the minimum delta 5% percentile indicates the number of IDs in the lower part of the data set. In this case, 7367045 *user IDs* and 14775 *item IDs* fall in the bottom 5% of the data set. The quantiles shown here indicate the influence of the hyperparameters MIN-COUNT-USER, MIN-COUNT-ITEM = 5, REMOVE-COMMON-USERS, REMOVE-COMMON-ITEMS = 0.1 | 0.05.

4.3.2. Data preprocessing

This section details the preprocessing of the *YooChoose* data set. Pre-processing can have a significant impact on the performance of the model [FC20]. It is important that the data preprocessing is done similarly as for the other data sets to achieve the best

possible comparability. The influence of different pre-processing on the model TGR will not be further investigated.

Edges

itemid (int64) The *item ID* is assigned a numeric value between 0 and the total number of unique items, called $i_{id} \in [0, |\mathcal{I}|]$.

session (int64) The *session ID* is assigned a numeric value ranging from 0 to the total number of sessions per user, denoted as $s_{id} \in [0, |s_{idu_{id}}|]$.

userId (int64) Since the data set does not contain a specific *user ID*, it is assumed that $u_{id} = s_{id}$.

timestamp (int64) The timestamp will be converted to Unix format.

4.4. Retailrocket

The data set *RetailRocket* is a public data set that contains anonymized information about user interactions with items on the website of an online retailer [Ret22]. It includes various events such as item views, purchases, and reviews. This data set is particularly valuable for the development and training of recommendation systems, as it provides insight into customer behavior with respect to a specific item [Ret22; Xu+19]. The data set *RetailRocket* provides a substantial amount of real-world data that can be used effectively to train and test recommendation algorithms. It is easily accessible and well suited for researchers and developers looking to improve recommendation systems in the e-commerce domain [Ret22].

The entire section is based on data from *RetailRocket* [Ret22], so the source is not repeated throughout.

4.4.1. Distribution of data

This section provides a detailed description of the *RetailRocket* data set, which is essential for the analysis and interpretation of the results in this thesis. To better understand the data set, the following subsubsections outline the structure of the data set.

Edges

The data set describing the edges has the following attributes.

visitorid (int64) This attribute provides a unique identifier for users, represented in this thesis as u_{id}. It enables tracking of user activity and behavior within the recommendation system.

itemid (int64) This attribute corresponds to the unique identifier of an item in the online store, denoted i_{id}. It allows you to associate visitor activity with the specific items they interact with.

event (object) This attribute indicates the type of event, which can be a view, purchase, or review. It is represented by values such as *addtocart*, *transaction*, or *view*.

timestamp (int64) The timestamp represents the moment when a user interacts with an item.

transactionid (float64) This attribute provides a unique ID for purchase of items on the site.

The Figure 4.5 shows the distribution of the interactions of users (u_{id}), items (i_{id}), and sessions (s_{id}). To keep the graph readable, the most frequent 0.01% are not shown. All statistical information is listed in Table 4.3.

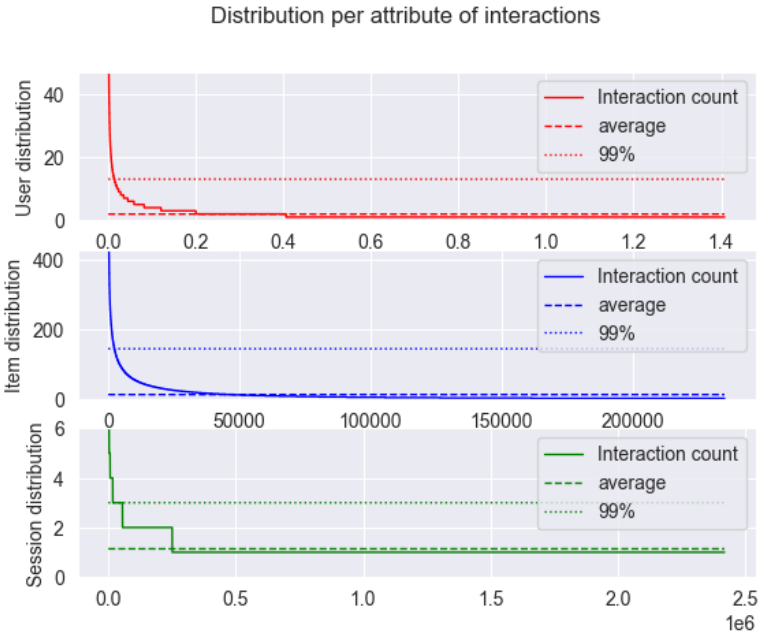


Figure 4.5.: Distribution of the data set *RetailRocket*

	$\lvert u_{id}\rvert$	$\lvert i_{id}\rvert$	$\lvert s_{id}\rvert$
count	1407580	235061	2417703
mean	1	11	1
min	1	1	1
99%	13	143	3
99.5%	19	209	4
max	7757	3412	45
Δ99%	12712	2343	17395
Δ99.5%	6610	1161	6785
Δmin2%	1001560	73609	2166955

Table 4.3.: Distribution of the data set *RetailRocket*

Table 4.3 shows the frequency distribution of $u_{id} \in \mathcal{E}$, $i_{id} \in \mathcal{E}$ and $s_{id} \in \mathcal{E}$ in the data set *Diginetica*. The variable $|u_{id}|$ indicates the frequency per unique *user ID* in the data set, $|i_{id}|$ indicates the frequency of unique *item IDs* and $|s_{id}|$ indicates the frequency of unique *session IDs*. For each variable, the quantiles of the minimum, maximum, mean, 99% and 99.5% are reported. The data set contains a total of 1407580 *user IDs*, 235061 *item IDs*, and 2417703 *session IDs*. On average, each user purchased 1 item, while each session contains 1 item.

The 99% quantile indicates that 99% of the users purchased a maximum of 13 items, while 99% of the sessions contain a maximum of 3 items. The 99.5% quantile indicates that 99.5% of the users interacted with a maximum of 19 items, while 99.5% of the sessions contain a maximum of 4 items. The maximum values for users, items, and sessions are 7757, 3412, and 45, respectively. The delta values show the differences between the values of the 99% and 99.5% quantiles and the maximum. These differences are important for identifying outliers in the data set, which can then be filtered and corrected with the appropriate hyperparameters. The delta values for the 99% quantile are 12712, 2343, and 17395, while the delta values for the 99.5% quantile are 6610, 1161, and 6785.

Finally, the minimum delta 2% indicates how many IDs are in the lower part of the data set. In this case, there are 1001560 *user IDs*, 73609 *item IDs*, and 2166955 *session IDs* in the bottom 2% of the data set. The quantiles presented here show the influence of the hyperparameters MIN-COUNT-USER, MIN-COUNT-ITEM = 2, REMOVE-COMMON-USERS,

REMOVE-COMMON-ITEMS = 0.1 | 0.05.

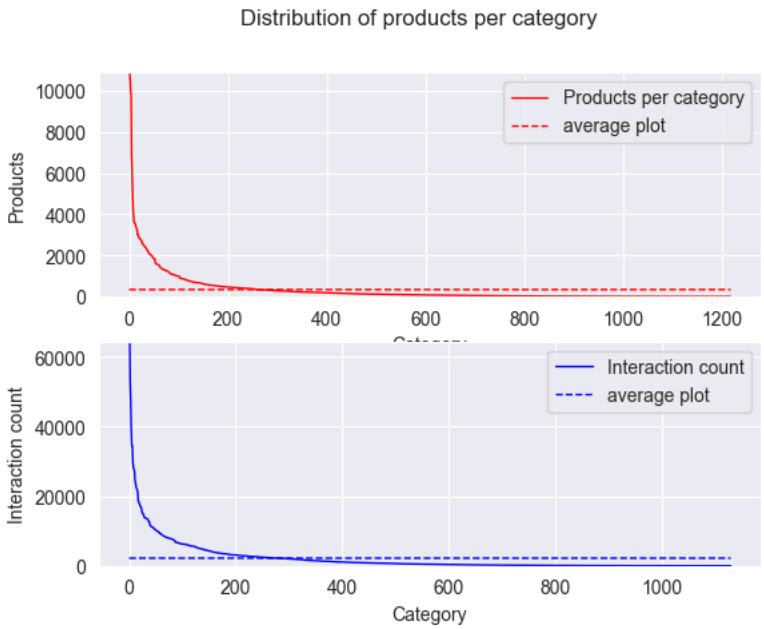


Figure 4.6.: Distribution of categories per item and interactions per category of the data set *RetailRocket*

It can also be seen in Figure 4.6 that many items belong to the same category, so it can be assumed that in this data set the jump of users to other items of different categories occurs less frequently. Consequently, the influence of the category change is negligible in this data set.

In Figure 4.6, an observation can be made concerning the distribution of items across different categories. It can be seen that a significant number of items belong to the same category. Consequently, it can be inferred that users within this particular data set have a relatively low tendency to switch to items from different categories. Therefore, the impact of category switching can be considered insignificant in the context of this data set.

This finding is of considerable importance because it provides valuable information on user behavior within the *RetailRocket* data set. By analyzing the distribution of the item and the frequency of category switching, a deeper understanding of user preferences and tendencies can be gained. The concentration of numerous items within a single category suggests a certain level of user preference for items within that category.

In addition, the low frequency of users switching between items in different categories

indicates a certain stability in user behavior. Users typically remain committed to items within the same category, which may indicate greater affinity or satisfaction with the offerings available in that particular category.

However, it is important to note that these conclusions are limited to the data set under investigation. User activity in other data sets or contexts may show different patterns of category switching and item exploration. Therefore, caution should be exercised when generalizing these findings to contexts outside the *RetailRocket* data set.

Items

The item data in this data set are straightforward and are presented in the following list.

itemid (int64) Represents a unique ID assigned to each item on the site. In this thesis, it is also called i_{id}.

property (object) Specifies the name of the property associated with the item.

timestamp (int64) Indicates the date and time that this attribute was posted.

value (object) Provides the value associated with the attribute.

There are special properties that are mentioned separately in the following list. It should be noted that the *Category ID* has already been used and analyzed above. However, the attribute *available* is not used in this thesis. This aspect could be explored in future studies to determine the usefulness of including items that cannot be purchased by the user.

Category ID (int64) represents a unique ID assigned to each category of *itemid* on the site.

available (bool) Specifies the attribute name that indicates the availability of the item.

4.4.2. Data preprocessing

This section details the preprocessing of the *RetailRocket* data set. Pre-processing can have a significant impact on the performance of the model [FC20]. It is important that the data preprocessing is done similarly as for the other data sets to achieve the best possible comparability. The influence of different preprocessing on the model TGR will not be further investigated.

Edges

visitorid (int64) The *user ID* is transformed from 0 to the total number of different users, denoted $u_{id} \in [0, |\mathcal{U}|]$.

itemid (int64) The *item ID* is transformed from 0 to the total number of different items, denoted as $i_{id} \in [0, |\mathcal{I}|]$.

session (int64) The session is created by combining u_{id} with the day of the timestamp, represented as $s_{id} = u_{id} + \text{day}$.

event (object) One-hot encoding is used to represent different types of events. This encoding is necessary because many machine learning algorithms cannot directly process categorical data.

timestamp (int64) The timestamp remains in the existing Unix format.

has_transaction (bool) A new attribute is created to indicate whether or not a transaction has occurred. This is determined by checking for the presence of a value in the *transactionid* attribute. Finally, the *transactionid* attribute is removed from the data set as it is no longer needed for prediction.

For the attributes u_{id}, i_{id}, and s_{id}, the considerations and transformations described in subsection 4.1.1 are applied.

Items

As items are provided in two separate data sets, they are merged and treated as a single data set for further processing. The *itemid* is considered within the same space, which is crucial for further processing since the IDs are used for identification purposes.

Next, the frequencies of each property (*property*) are computed and only those with more than 1300 occurrences are selected. This step is used to reduce the size of the data and eliminate unnecessary information.

Finally, all text columns are converted to numeric values for further processing. This is achieved using the LabelEncoder, which assigns numeric values to text-based data. Finally, all numeric columns are standardized to ensure that they are on a consistent scale, facilitating meaningful comparisons and analysis. To categorize the items, the i_{id} is transformed into $i_{id} \in [0, |\mathcal{I}|]$, the same as $i_{id} \in \mathcal{E}$.

4.5. Refurbed

The data set *Refurbed* is not publicly accessible. This dataset contains non-anonymized information about users' interactions with items on an online shopping website. Data includes interactions, detailed item, and user data. The data set is particularly useful for developing and training recommendation systems, as it provides information about users, items, and customer behavior in relation to specific items.

The data set *Refurbed* provides a substantial amount of real-world data that can be used to train and test the algorithms of the recommendation system.

4.5.1. Distribution of data

In this section, the *Refurbed* dataset is described in detail. The explanation and processing of the data set is of central importance for interpreting and discussing the results. Therefore, they play a crucial role in subsequent experiments using this data set. To enhance understanding of the data, this section provides a detailed description of the structure of the data set. First, the interaction data will be described, which will be referred to as *Edges* in the following, while the item data will be referred to as *Items*.

Edges

The data set describing the interactions on the website consists of multiple files with varying attributes. For the purposes of this thesis, the data set *interaction_type_click_merged* will be focused on, as it contains the relevant interaction information. The other data set that describes the purchases will not be used further in this thesis because it is beyond the scope of this study. All IDs are only available within the dataset and do not allow the use of user-related data beyond that. The attributes of the *interaction_type_click_merged* data set are as follows.

fullVisitorId (int64) A unique identifier for the user, referred to as u_{id} in this thesis. This attribute allows tracking the user's activity and behavior in the recommendation system.

ga_sessions_id (int64) A unique identifier for the visitor's Google Analytics session. This attribute can provide information about the visitor's behavior during a session and assist in generating personalized recommendations based on their interactions. This attribute can provide information on the user's behavior within a session and

help generate personalized recommendations based on their interactions during the session. It is referenced in this thesis as s_{id} .

product_id (int64) This attribute, referred to as i_{id}, is the unique identifier of an item in the online store. This attribute enables the model to associate visitors' activities with the items they interact with.

instance_id (int64) A unique identifier for a particular instance of an item. This attribute can be used to manage different instances of an item. For example, if an attribute changes, it can be tracked which instance the user interacted with.

hits_timestamp (object) The timestamp of a user's interaction with an item. This attribute, referenced in this thesis as $t^{(u,i)}$, is used to analyze the temporal pattern of user behavior and generate recommendations based on current interactions or past preferences and is referenced in this thesis as $t^{(u,i)}$.

date_first_visit (object) This field represents the date when the user first visited the online store. This attribute could help the model to recommend suitable items.

price (object) The price of a specific item. This attribute could enable the model to take into account the user's preferred price range.

quality (object) A rating of the condition of the item that is being interacted with. This attribute could enable the model to take into account the visitor's preferred item condition.

delivery_days_min (int64) The minimum number of days required to deliver an item. This attribute allows the recommendation system to generate recommendations based on the user's delivery time preference.

delivery_days_max (int64) The maximum number of days required to deliver an item. Similar to *delivery_days_min*, this attribute allows recommendations to be generated based on the user's delivery time preference.

warranty (int64) The warranty period for a particular item. This attribute can be used in the recommendation calculation to prioritize items with longer warranties or to make special offers for items with expiring warranties.

page_country (object) The country where the visited page is located. This attribute allows the recommendation system to generate country-specific recommendations or local offers.

country (object) The user's country. This attribute can be used to consider regional variations in the user's preferences and generate suitable recommendations.

city (object) The city where the user is from. Similarly to *country*, this attribute can be used to generate recommendations based on city preferences or specific offers in certain cities.

channel (object) The channel through which the visitor accessed the online store (e.g., direct search, social media, referral). This attribute allows the recommendation system to differentiate between various channels and customize recommendations based on the user's behavior on each channel.

device_browser (object) The user's web browser. This attribute can be used to customize recommendations based on the technical capabilities or preferences of the browser.

device_category (object) The category of device the user is using (e.g. desktop, mobile, tablet). This attribute allows the recommendation system to optimize recommendations based on device category and customize the user experience for different device types.

unique_products_per_user (int64) The number of unique items viewed by the user. This attribute can be used to assess the diversity of the user's interests and generate recommendations for similar or complementary items.

second_spent_on_page (float64) The amount of time (in seconds) the user spent on a particular page. This attribute can be used to measure the user's interest in a specific page or item. It allows the recommendation system to evaluate the relevance and importance of a page to the visitor and adjust recommendations based on the visitor's engagement. Pages where the user has spent more time can be considered more important and receive appropriate recommendations.

The Figure 4.7 shows the distribution of the interactions of users (u_{id}), items (i_{id}), and sessions (s_{id}). To maintain readability, the graph does not display the most frequent 0.01%. All statistical information is listed in Table 4.4.

Table 4.4 represents the frequency distribution of $u_{id} \in \mathcal{E} i_{id} \in \mathcal{E} s_{id} \in \mathcal{E}$ in the data set *Refurbed*. The variables $|u_{id}|$, $|i_{id}|$, and $|s_{id}|$ represent the frequencies of unique *user IDs*, *item IDs*, and *session IDs*, respectively. Quantile values for minimum, maximum, mean, 99%, and 99.5% are provided for each variable.

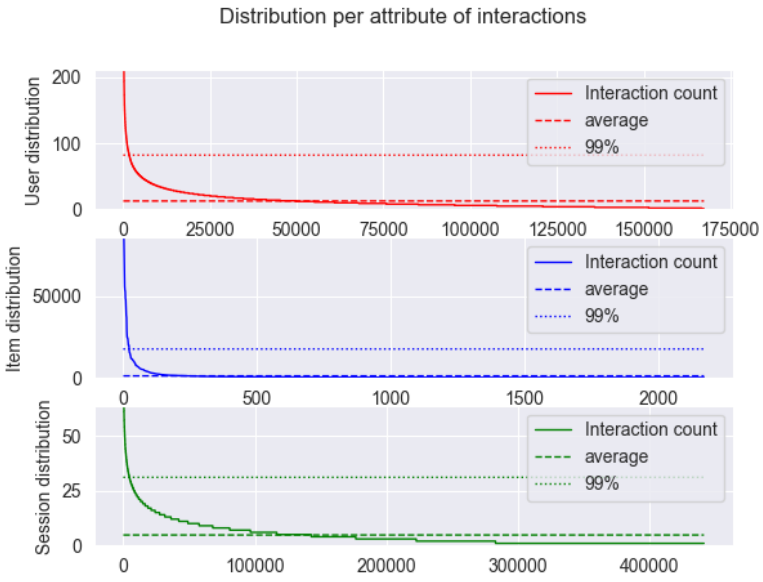


Figure 4.7.: Distribution of the data set *Refurbed*

The data set contains a total of 167379 u_{id}, 2171 i_{id}, and 441360 s_{id}. On average, each user interacted with 12.65 items, while each session contained 4.79 items. The 99% quantile indicates that 99% of users purchased a maximum of 23 items and 99% of sessions contained a maximum of 31 items. For the 99.5% quantile, the corresponding values are 111 items for users and 40 items for sessions. The observed maximum values were 1597, 189917, and 204 for user, item, and *session IDs*, respectively.

The Δ values represent the differences between the 99% and 99.5% quantile values and the maximum. These differences play a critical role in identifying outliers in the data set, which can be filtered and corrected with appropriate hyperparameters. Specifically, the Δ values for the 99% quantile are 1666, 22, and 4382, while the Δ values for the 99.5% quantile are 817, 11, and 2069.

Furthermore, the minimum delta 5% indicates the number of IDs that fall in the bottom 5% of the data set. In this case, 46457 *user IDs*, 184 *item IDs*, and 298580 *session IDs* are in the bottom 5%. The quantiles given show the effect of the hyperparameters MIN-COUNT-USER, MIN-COUNT-ITEM (set to 5), REMOVE-COMMON-USERS, REMOVE-COMMON-ITEMS (set to 0.1 and 0.05, respectively).

In Figure 4.8 the distribution of categories per item and the distribution of interactions by category. In this figure, it is evident that multiple elements belong to the same category. It can be concluded that, in these data, the application of elements from other

	$\lvert u_{id} \rvert$	$\lvert i_{id} \rvert$	$\lvert s_{id} \rvert$
count	167379	2171	441360
mean	12.65	975.78	4.79
min	1	1	1
99%	82	17474	31
99.5%	111	38844	40
max	1597	189917	204
Δ99%	1666	22	4382
Δ99.5%	817	11	2069
Δmin5%	46457	184	298580

Table 4.4.: distribution of the data set *Refurbed*

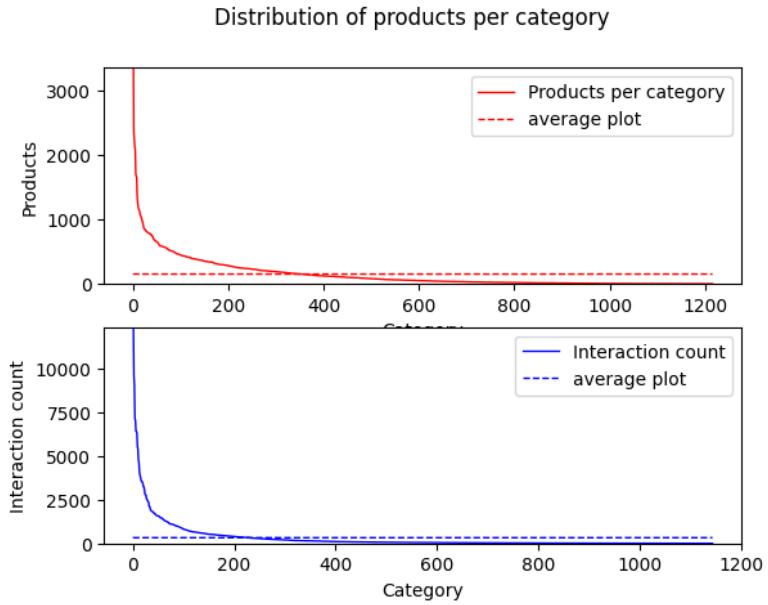


Figure 4.8.: Distribution of categories per item and interactions per category in the data set *Refurbed*

categories is less frequent. Therefore, the influence of category switching is negligible in this data set. In addition, in this data set, there is no statistically significant correlation between the interactions. The correlation between time and i_{id} is 0.0067.

Items

The item data in this data set is simple and is presented in the following list.

instance_id (int64) This is equivalent to the *instance_id* mentioned above.

product_id (int64) This is equivalent to *product_id* as shown above.

instance_name (object) The name of the specific instance of the item.

product_name (object) The general name of the item, regardless of the instance.

scope_of_delivery (object) A description or enumeration of the included elements or parts of the item.

reference_price (float64) The price used as a reference for the item.

brand (object) The manufacturer of the item.

category (object) The category or classification to which the item belongs. This attribute allows similar items to be grouped in the same category.

attribute_id (int64) A unique identifier for a specific attribute of the item. This attribute can be used to distinguish different properties or characteristics of the item.

attribute_name (object) The name or description of the attribute. This attribute indicates the property or characteristic.

att_value (object) The value or property of the attribute. This attribute contains the specific value or expression of the corresponding property or characteristic.

Users

fullvisitorId (int64) Corresponds to *fullvisitorId* presented above.

date_first_visit (float64) The date of the first visit.

date_last_visit (float64) The date of the last visit.

first_is_mobile (int64) A flag indicating whether the first visit was from a mobile device.

Attribute	Values	Count
Farbe	transparent, schwarz, rot, ...	37892
Hinweis	text	31645
Abmessungen	358 x 16.2 x 246 mm, 214.9 x 280.6 x 6.4 mm	30649
Anschlüsse	4 x Thunderbolt 3, Audio in/out, USB-C 4.0, ...	30179
Betriebssystem	macOS, iPadOS, ab Android 10m ...	29806
Speicherplatz	8, 4, 0.5, 128, ...	29483

Table 4.5.: The 5 most frequently used attributes with sample values and the frequency of these attributes from the data set *Refurbed*

first_device_type (object) The type of the first device, such as smartphone, tablet, or desktop. This can be used to customize recommendations based on screen size or technical capabilities.

first_device_os (object) The operating system of the first device. This could help the model to make appropriate recommendations, e.g. it is obvious that Apple customers stay in the ecosystem.

first_channel (object) The first channel through which the visitor arrived at the site.

country (object) The country of the user. Allows the model to generate country-specific recommendations.

first_browser (object) The first web browser used.

first_medium (object) The first medium through which the visitor arrived at the site.

first_source (object) The first source from which the visitor came to the website (e.g. Google, Facebook).

4.5.2. Data preprocessing

This section details the preprocessing of the *Refurbed* data set. Pre-processing can have a significant impact on the performance of the model [FC20]. It is important that the data preprocessing is done similarly as for the other data sets to achieve the best possible comparability. The influence of different preprocessing on the model TGR will not be further investigated.

Edges

As this data set has extensive information, the data representing the events will only be described in part; the data that are crucial from the author's point of view and that make the data set particularly valuable for this thesis will be described.

One particularly interesting feature of this data set is that, unlike the other data sets used in this thesis, it contains both device and location data.

Items

The most important properties of the items are presented below. This data set is characterized by the availability of a comprehensive set of properties for each item. The scope of delivery and manufacturer data are shown in particular.

itemid (int64) a unique ID for each item on the website.

property (object) A set of data that precisely describes the item.

name (string) The name of the item

brand (string) specifies the manufacturer

User

In the following, the attributes of the users are presented. In addition to the events, the users also have geo- and device data, which distinguishes this data set in that it also has detailed user data in addition to item and event data.

userId (int64) a unique ID for each user on the web site.

timestamp (int64) specifies the first and last timestamp of the user.

geodata (object) a set of data related to the user's location.

objectdata (object) A set of data related to the user's device.

4.6. Conclusion

This section provides a comprehensive synthesis and careful analysis of the four data sets. Each data set illuminates particular aspects of evaluating TGR. The respective

significance and unique contribution of each data set to the investigation and evaluation of TGR is considered and carefully delineated. The data set *RetailRocket* is encapsulated in subsection 4.6.1, *YooChoose* in subsection 4.6.2, *Diginetica* in subsection 4.6.3, and *Refurbed* in subsection 4.6.4.

Data set	**User features**	**Has sequences**	**Item features**	**Has category**
Refurbed	Yes	Yes	Yes	Yes
YooChoose	No	No	No	No
Diginetica	No	Yes	Yes	Yes
RetailRocket	No	Yes	Yes	Yes

Table 4.6.: Overview of data set features. The **User Features** column indicates the inclusion of user features. The **Has Sequences** column indicates the presence of sequential data in the data set. The **Item Features** column reflects the presence of distinct attributes associated with items. Finally, the column **Has Category** indicates the categorization status of the items.

The Table 4.6 provides a systematic and comparative analysis of the data sets used with respect to the availability of specific features. The **User features** column indicates whether a data set contains certain attributes or user characteristics. An affirmative *Yes* in this column indicates the presence of such features, while an *No* indicates their absence. In addition, the **Has sequences** column indicates whether the data set contains sequential data. The Has **Item features** column indicates the presence of specific attributes of the items or items contained in the records. Finally, the **Has category** column indicates whether items are categorized.

Refurbed stands out for completeness, scoring positive in all attribute categories. *YooChoose*, on the other hand, has a negative score in all categories, indicating a rudimentary structure or lack of detailed information. Interestingly, the data sets *Diginetica* and *RetailRocket* have similar characteristics: Both do not contain user features but provide sequential data, item features, and category information.

The different availability of features allows the usefulness of each feature to be tested and evaluated in different ways.

4.6.1. Retailrocket

RetailRocket is a large and diverse data set in the field of e-commerce. It contains 2756101 interactions involving 1407580 anonymized users and 235061 different items over four

and a half months. As a result, this data set encapsulates a wide range of interactions and provides deep insights into user behavior in an e-commerce environment [Ret22].

This data set delineates three distinct types of events, specifically *view*, *addtocart*, and *transaction*. These event classifications provide a comprehensive understanding of users' specific actions on the platform. However, these event categories are not considered further in this thesis. Although they have the potential to provide invaluable information on user behavior, the focus of this thesis is not on this aspect of the data set [Ret22].

In addition to event types, *RetailRocket* provides a rich set of attributes for each item, including categories and pricing information. This variety of attributes allows for a sophisticated analysis of the characteristics of the items with which users interact, potentially providing valuable information on user preferences and behaviors [Ret22].

This data set has significant value for the evaluation conducted in this study. It could reveal whether user information is critical to generate recommendations or whether raw interaction data is sufficient to generate high-quality recommendations. Furthermore, it could reveal whether sessions must be explicitly present or whether their programmatic synthesis is sufficient to generate explicit session recommendations. This aspect becomes particularly interesting compared to *Refurbed*, as the latter lacks certain information provided by the former.

4.6.2. YooChoose

YooChoose, initially released in conjunction with the 2015 RecSys Challenge, represents a robust and multifaceted repository of information. Capture click behavior in 9249729 sessions in an e-commerce marketplace over six months. With an extensive data set of 33003944 clicks and purchases associated with 52739 different items, this data set provides a comprehensive view of user interactions within an e-commerce environment [Yoo22].

Unlike the other three data sets, *YooChoose* does not include auxiliary attributes associated with individual items. A notable feature of this data set is its exclusive emphasis on session data, excluding user and product data. Additionally, purchase data is not used in this particular study [Yoo22].

This unique prioritization of session data makes *YooChoose* exceptionally valuable for the evaluation conducted in this thesis. It allows the investigation of whether user and product information is critical to recommendation generation or whether raw interaction data are sufficient to generate high-quality recommendations.

There are other intriguing research opportunities compared to the *Refurbed* data set, which is rich in information missing from *YooChoose*. Comparative analysis between these data sets allows for a deeper understanding of the impact of different types of information on the quality of recommendations.

4.6.3. Diginetica

Diginetica, originally derived from the RecSys Challenge 2016, is another comprehensive data source. While it provides extensive information, it is comparatively smaller than the other two data sets *RetailRocket* and *YooChoose*. Covers the behavior of a relatively modest user base of 232816 individuals. In addition, this data set focuses primarily on click data, although it is enriched with additional data elements such as product categories and pricing details. The data set encapsulates data from 22816 users who interact with 184047 different items [DIG16].

Despite its reduced size, the *Diginetica* data set is of significant value for the evaluation conducted in this study. It could provide further insight into whether user information is central to generating recommendations. A particularly noteworthy feature of this data set is the inclusion of session information. This allows testing and analytical examination of the connections between different sessions. This feature allows for a deeper exploration of the dynamics and structures of user sessions, which could contribute to a more sophisticated understanding of user behavior and its implications for recommendation systems.

4.6.4. Refurbed

Refurbed is another comprehensive data set, although it is not publicly available, which confers a degree of exclusivity. This data set encapsulates 2118425 interactions conducted by a user pool of 167379 individuals. These interactions were carried out over 441360 sessions involving 2171 unique items. A unique feature of this data set includes product and user characteristics.

The contribution of this data set to the evaluation conducted in this study is significant. Encapsulation of product and user information provides a unique opportunity to assess potential benefits while evaluating these features. Another intriguing aspect of this data set is its smaller volume of items compared to the other data sets. This unique feature could positively impact the metrics, as the likelihood of suggesting the correct product could theoretically increase due to a smaller pool of items. This aspect may lend itself to

a more in-depth exploration of how the total number of items can affect the efficiency and accuracy of recommendation systems.

5. Experiments

This chapter evaluates TGR and measures its prediction quality using various experiments. Several datasets and metrics are used to evaluate the quality of the recommendations. The used data sets are presented in chapter 4, and the necessary pre-processing steps are explained. The metrics used are defined in subsection 2.5.2.

The experiments presented in section 5.3, section 5.4, section 5.5, and section 5.6 were performed on the data sets *Refurbed*, *RetailRocket*, *YooChoose*, and *Diginetica*. These sections evaluate various aspects of the network controlled by the hyperparameters specified in subsection 3.7.1. The results of the evaluation are discussed and interpreted. In this context, the possible limitations and constraints of the experiments performed are also addressed. The results of this chapter will help to deepen the understanding of how the model works and its potential to enhance the quality of recommendations.

For each data set, DATA-SPLIT-RATIO = 0.6, so 60% is used for training, 20% for testing, and 20% for validation. The edges are sorted chronologically.

5.1. Experiment Setups

In this section, the basic aspects of the various experiments are presented. In this thesis, five experiments are performed, represented by the letters A-B. Each one of the experiments has a subsection of this section below, in which the setup is presented. Each model for the experiments is trained with 100 epochs and the Adam optimization algorithm with a learning rate of $3e-4$.

5.1.1. Experiment A: Effect of run modes

The purpose of *Experiment A* is to investigate the impact of different RUN-MODE on the results of TGR. The effects and significance of RUN-MODE are described in subsection 3.7.1. The hyperparameters listed in Table 5.1 apply to this experiment. Otherwise, the hyperparameters from chapter subsection 3.7.1 apply. In this experiment, the network is trained using the three different RUN-MODE.

Hyperparameter	Values
MIN-COUNT-USER	$5 \in \mathbb{N}$
MIN-COUNT-ITEM	$5 \in \mathbb{N}$
REMOVE-COMMON-USERS	$0.05 \in \mathbb{R}$
REMOVE-COMMON-ITEMS	$0.05 \in \mathbb{R}$
NEGATIVE-SAMPLER	most similar
LEARNING-RATE	$3e-4$

Table 5.1.: Hyperparameter setup for *Experiment A*

The experimental results of *Experiment A* are subjected to a comprehensive analysis to evaluate the impact of the different RUN-MODE on the performance of the recommendation system. It is expected that SESSION performs better than STRUCTURAL, because SESSION performs an explicit calculation per user, which should positively impact the accuracy of session recommendations. In data sets containing sessions (e.g., *Refurbed* and *Diginetica*), HISTORY is expected to give better results because HISTORY can better model the relationships between different sessions. On the other hand, for data sets that do not have this edge feature, no significant improvement is expected. The exact results can be found in the corresponding subsections of the data sets tested in this chapter.

5.1.2. Experiment B: Impact of run modes with negative sampling

The purpose of *Experiment B* is to examine the effects of different NEGATIVE-SAMPLER associated run modes on TGR results. The effects and meaning of RUN-MODE are described in *Experiment A*. The explanation of NEGATIVE-SAMPLER is described in subsection 3.7.1. For this experiment, the hyperparameters listed in Table 5.1 apply, with the modification that NEGATIVE-SAMPLER is examined in this experiment. In this experiment, all considerations of *Experiment A*.

5.1.3. Experiment C: Effect of category-shift

Experiment C investigates the effects of USE-CATEGORY-SWITCH. The effects of USE-CATEGORY-SWITCH are described in subsection 3.7.1. For this experiment, the hyperparameters listed in Table 5.2 apply.

Assuming that users who regularly switch between item categories show significant behavioral patterns, it is expected that the function controlled by the USE-CATEGORY-

Hyperparameter	Values
MIN-COUNT-USER	$5 \in \mathbb{N}$
MIN-COUNT-ITEM	$5 \in \mathbb{N}$
REMOVE-COMMON-USERS	$0.05 \in \mathbb{R}$
REMOVE-COMMON-ITEMS	$0.05 \in \mathbb{R}$
NEGATIVE-SAMPLER	most similar
LEARNING-RATE	$3e-4$
RUN-MODE	SESSION
USE-CATEGORY-SWITCH	0

Table 5.2.: Hyperparameter setup for *Experiment C*

SWITCH parameter can be used to target this group of users. This could potentially lead to an optimization of the quality of the prediction models.

5.1.4. Experiment D: Effect of temporally stored session embeddings

The purpose of *Experiment D* is to investigate the effects of USE-SESSION-TIME-SAVE. The effects of USE-SESSION-TIME-SAVE are described in subsection 3.7.1. The hyperparameters listed in Table 5.3 apply to this experiment.

Assuming that a user's sessions in the past show a significant deviation in the behavioral pattern, it is expected that the use of the function controlled by the USE-SESSION-TIME-SAVE parameter will allow for an interruption of these historical sessions. A reduced weighting of historical sessions could lead to an optimization of the quality of the prediction, as the current session is given more prominence.

5.1.5. Experiment E: Effect of sequence length

Experiment E examines the effects of varying sequence lengths and the effects of filtering out the most frequently used nodes in the graph. This experiment uses the hyperparameters specified in Table 5.4. *Experiment E* is based on the best runs of previous experiments on the corresponding data set and, therefore, has different hyperparameters depending on the dataset, so $\cdot$ is considered a placeholder.

It is expected that shorter sequence lengths will have a negative impact on the results. Similarly, removing more of the most frequently used nodes is also expected to have

Hyperparameter	Values
MIN-COUNT-USER	$5 \in \mathbb{N}$
MIN-COUNT-ITEM	$5 \in \mathbb{N}$
REMOVE-COMMON-USERS	$0.05 \in \mathbb{R}$
REMOVE-COMMON-ITEMS	$0.05 \in \mathbb{R}$
NEGATIVE-SAMPLER	most similar
LEARNING-RATE	$3e-4$
RUN-MODE	SESSION
USE-CATEGORY-SWITCH	0
USE-SESSION-TIME-SAVE	0

Table 5.3.: Hyperparameter setup for *Experiment D*

Hyperparameter	Values
MIN-COUNT-USER	$5 \in\in \{5, 10\} \in \mathbb{N}$
MIN-COUNT-ITEM	$5 \in\in \{5, 10\} \in \mathbb{N}$
REMOVE-COMMON-USERS	$0.05 \in \{0.05, 0.1\} \in \mathbb{R}$
REMOVE-COMMON-ITEMS	$0.05 \in \{0.05, 0.1\} \in \mathbb{R}$
NEGATIVE-SAMPLER	·
LEARNING-RATE	$3e-4$
RUN-MODE	·
USE-CATEGORY-SWITCH	·
USE-SESSION-TIME-SAVE	·

Table 5.4.: Hyperparameter setup for *Experiment E*

Run	E	Min nodes	Max nodes	RUN-MODE	NS	CS	TS
1	A/B	5	0.05	STR	MOST-SIMILAR	1	1
2	A/B	5	0.05	SES	MOST-SIMILAR	1	1
3	A/B	5	0.05	HIS	MOST-SIMILAR	1	1
4	B	5	0.05	STR	RANDOM	1	1
5	B	5	0.05	SES	RANDOM	1	1
6	B	5	0.05	HIS	RANDOM	1	1
7	B	5	0.05	STR	LESS-SIMILAR	1	1
8	B	5	0.05	SES	LESS-SIMILAR	1	1
9	B	5	0.05	HIS	LESS-SIMILAR	1	1
10	C	5	0.05	SES	MOST-SIMILAR	0	1
11	D	5	0.05	HIS	MOST-SIMILAR	0	0
13	D	5	0.05	HIS	MOST-SIMILAR	1	0
14	D	5	0.05	HIS	LESS-SIMILAR	0	0
15	E	5	0.10	·	·	·	·
16	E	3	0.05	·	·	·	·

Table 5.5.: Hyperparameter setup

a negative effect. Patterns in longer sequences are easier to detect, and it is easier to predict the most frequent elements than those with few interactions [Wu+20a; Gao+21].

5.1.6. Experiment setup overview

The experimental setup of the conducted experiments is presented as an overview in Table 5.5. This table contains columns numbered by the experiment run (Run), followed by the specific hyperparameter and the corresponding experiment (E). This table is used as a reference for further analysis. The detailed documentation of the hyperparameters in the previous sections and the analyses based on them ensure the reproducibility of the experiments and allow conclusions to be drawn about the performance of the TGR model presented in this thesis based on the hyperparameters.

Due to space limitations, the headings in Table 5.5 have been abbreviated as shown below: *Min Nodes* and *Max Nodes* represent the values of MIN-COUNT-USER = MIN-COUNT-ITEM = *Min Nodes* and REMOVE-COMMON-USERS = REMOVE-COMMON-ITEMS = *Max Nodes*. Furthermore, RUN-MODE STR represents STRUCTURAL, SES represents

SESSION, and HIS represents HISTORY. Additionally, *CS* applies to USE-CATEGORY-SWITCH, *TS* applies to USE-SESSION-TIME-SAVE, and *LESS*, *MOST*, and *Random* apply to the corresponding NEGATIVE-SAMPLER (*NS*). This table provides an overview since all other sections depend on it. Run 12 is a special case and is similar to run 11 in all hyperparameters presented here. Run 12 is explained in the corresponding section. *Experiment E* is based on the best runs of previous experiments in the corresponding data set and therefore has different hyperparameters depending on the data set.

5.2. Baseline Models

The models presented in this section will be comparison models in the following experiments. Each model offers interesting approaches that make the comparison meaningful [Hou+22].

FPMC

Factorized Personalized Markov Chains (FPMC) is a sequential recommendation model that captures the dynamics of user actions using factorized Markov chains [RFS10]. This model uses the Markov property to depict the transition probabilities between user actions. By integrating user and item attributes, FPMC enables personalized recommendations that consider both the sequence of actions and the user's preferences [RFS10]. The factorized representation used by FPMC enables the efficient computation of probability distributions, facilitating the generation of scalable recommendations [RFS10].

GRU4Rec

Gated Recurrent Unit for Recommendation Systems (GRU4Rec) is a recommendation model that uses RNNs to identify sequential patterns in user actions [Hid+16]. It uses GRU, described in subsection 2.2.9, to determine the importance and relevance of past actions in predicting future recommendations [Hid+16]. The GRU4Rec model uses the GRU gating mechanism to selectively regulate the influence of previous actions during the recommendation process. By capturing user behavior through sequential data, GRU4Rec is able to generate accurate and personalized recommendations [Hid+16].

NARM

Neural Attentive Recommendation Model (NARM) is a recommendation model that applies attention mechanisms to learn sequential patterns of user interactions and the importance of individual items in those interactions [Li+17]. NARM uses attention layers to prioritize relevant items and adjust recommendations accordingly [Li+17]. Attention layers are described in subsection 2.2.7. By integrating attention and sequence modeling, NARM can generate accurate and contextually relevant recommendations for users [Li+17].

SR-GNN

Session-based Recommendation with Graph Neural Networks (SR-GNN) is a recommendation model that uses session graphs [Wu+19]. Session graphs are described in section 2.2 and represent interactions between users and items as a graph. By incorporating the session context and using the graph structure, SR-GNN effectively captures the relevance and relationships between items [Wu+19]. The use of GNN enables SR-GNN to capture complex patterns and dependencies within the data, facilitating the generation of personalized recommendations [Wu+19].

NISER+

Normalized Item and Session Representations to Handle Popularity Bias (NISER+) is a recommendation model that addresses popularity bias in session-based recommendation systems [Gup+21]. This model uses the normalized item and user session representations to mitigate the effects of popularity [Gup+21]. By incorporating appropriate weights and adjustments into the model equations, it effectively accounts for less popular items. The evaluation results demonstrate the effectiveness of NISER+ in significantly reducing popularity bias [Gup+21].

LESSR

Graph-less Neural Networks (LESSR) is a recommendation model that addresses the problem of information loss when using GNNs for session-based recommendations [CW20]. GNNs, as described in section 2.2, suffer from the problem of information loss during session encoding. The model introduces a lossless coding method and a GRU based edge-order preserving aggregation layer [CW20] to address this challenge. Furthermore,

the model addresses the problem of ineffective capture of long-range dependencies by incorporating a graph shortcut attention layer [CW20]. This layer facilitates information propagation along shortcut links, enabling the model to capture long-range dependencies more effectively [CW20].

SGNN-HH

Star Graph Neural Networks (SGNN-HN) method utilizes a *star* GNN, as explained in section 2.2, to offer recommendations in session-based scenarios [Pan+20]. Unlike conventional approaches that focus solely on the sequential ordering of items for session recommendations and overlook complex transition relationships, this approach addresses this issue, employing SGNN-HN to capture the transitions between items within an ongoing session [Pan+20]. The resulting aggregated embeddings effectively capture the user's preferences to predict items [Pan+20].

SASRec

Self-Attentive Sequential Recommendation (SASRec) is a recommendation model designed to improve the accuracy of predictions in the context of sequential user actions [KM18]. This model uses self-aware mechanisms to identify and analyze dependencies and patterns within a sequence of user actions. By effectively capturing the relationships between different user actions, SASRec can adapt the recommendations accordingly [KM18]. The key innovation in SASRec is the use of attention mechanisms, which allow it to assign different levels of importance to past actions when making future recommendations [KM18]. By incorporating self-attention, SASRec can dynamically adjust its recommendations based on the discovered dependencies, resulting in accurate and personalized suggestions [KM18]. Attention layers are described in subsection 2.2.7.

GC-SAN

Graph Convolutional Self-Attention Network (GC-SAN) is a recommendation model that combines GCNs [Xu+19] with self-attention mechanisms to improve accuracy [Xu+19]. GCNs are explained in Equation 2.4 and. This model effectively captures structured information from user actions and adapts recommendations accordingly [Xu+19]. In GC-SAN, GCNs are used to represent the connections between users and objects in a graph. Using GCNs, the model can analyze the intricate connections between users and items, thus capturing the intricate patterns and dependencies present in the data [Xu+19].

In addition, GC-SAN incorporates self-attention mechanisms to assess the relevance and importance of individual items., which allows the model to focus on significant items and consider their influence on the recommendation process [Xu+19].

CL4Rec

Contrastive Learning for Sequential Recommendation (CL4Rec) is a recommendation model that uses the contrastive learning approach to capture the relevance of items within sequences effectively [Xie+21b]. This model focuses specifically on understanding the interactions between users and items within sessions [Xie+21b]. It achieves this by employing contrastive learning, which enables the identification of similar items within the same session while distinguishing dissimilar items across sessions. CL4Rec effectively learns and utilizes item embeddings to generate precise, personalized recommendations. Using contrastive learning, CL4Rec significantly improves its ability to generate accurate and personalized recommendations [Xie+21b].

CORE

Simple and Effective Session-Based Recommendation within Consistent Representation Space (CORE) is a recommendation model that addresses the problem of inconsistent predictions in session-based recommendation systems [Hou+22]. This model achieves this by integrating sessions and articles into a unified representation space and using a representation-consistent encoder that uses a linear combination of input article embeddings to obtain session embeddings [Hou+22]. Furthermore, a robust distance measurement technique is used to mitigate overfitting of the embeddings in the unified representation space. The efficacy and efficiency of the proposed approach are demonstrated through extensive experiments conducted on five real-world datasets [Hou+22].

5.3. Refurbed

This chapter describes the experiments and their results with the *Refurbed* dataset described in section 4.5. First, the effect of the hierarchical architecture on the network prediction accuracy is investigated. Then, different parameters and configurations of the model are examined to measure their impact on the quality of the recommendation.

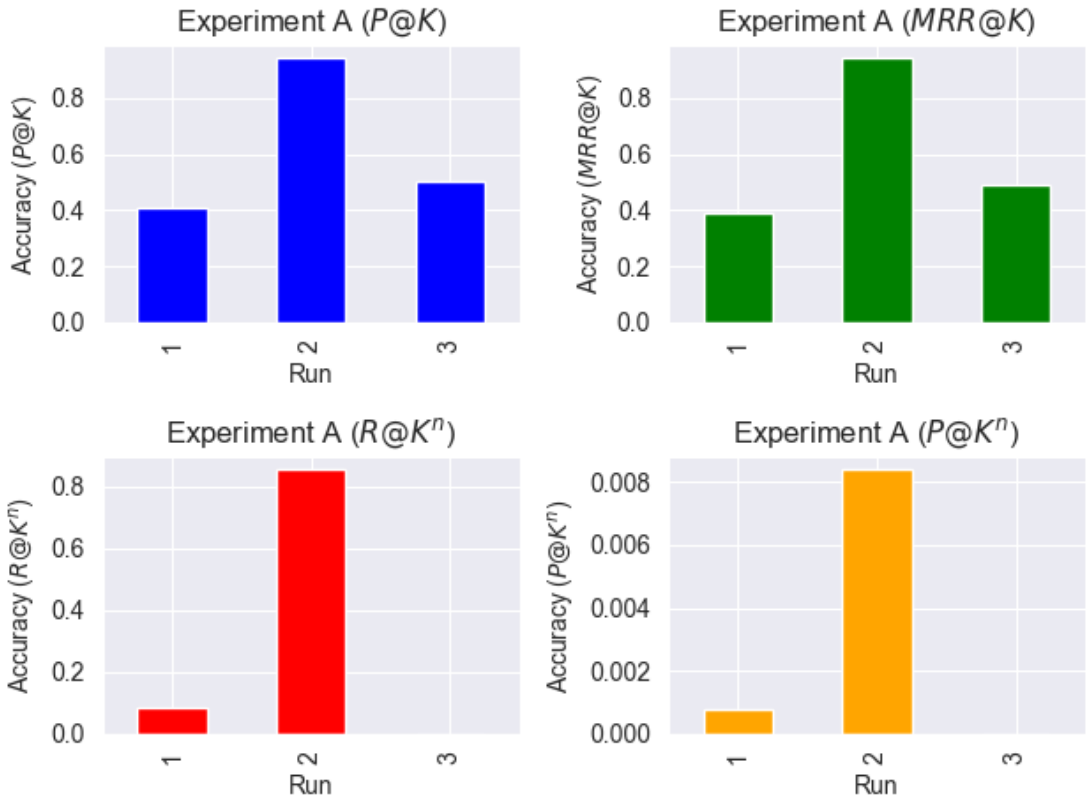


Figure 5.1.: Results of *Experiment A* for the data set *Refurbed* with $\mathcal{K} = 10$. Each bar represents a run of the experiment performed under different specifications of the hyperparameter RUN-MODE. The first, second, and third experimental runs are associated with STRUCTURAL, SESSION, and HISTORY, respectively. This graphical representation facilitates an analytical inspection of the effects induced by variations in the RUN-MODE hyperparameter.

5.3.1. Experiment A

In Figure 5.1, the results of *Experiment A* are shown for the data set *Refurbed* with $\mathcal{K} = 10$. The experimental setup is described in subsection 5.1.1. The metrics used are defined in subsection 2.5.2. All results are available in Table A.1.

The results show that RUN-MODE with the parameter SESSION in combination with negative sampling using MOST-SIMILAR achieves the best results. The recommendation system achieves an outstanding $P@K$ value of 0.9429 and an almost identical $MRR@K$ value of 0.9420. These high values indicate that the system can generate a significant

percentage of correct recommendations within the top $\mathcal{K}$ recommendations and that the ranking of correct recommendations is nearly optimal.

In contrast, the other two parameters for RUN-MODE, namely STRUCTURAL and HISTORY, have lower scores. The STRUCTURAL mode results in a $P@K$ value of 0.4061 and a $MRR@K$ value of 0.3864, while the HISTORY mode results in a $P@K$ value of 0.5034 and a $MRR@K$ value of 0.4870. These values are significantly lower than those of the parameter SESSION, suggesting that the recommendations in these settings RUN-MODE are less precise and less relevant to users.

This is also evident in the metrics $R@K^n$, where the model with STRUCTURAL achieves a value of 0.0832, and $P@K^n$, with a value of 0.0008. These values indicate that the recommendation system in this mode has difficulty accurately identifying and recommending the target item based on negative examples.

For SESSION, a significant performance improvement is observed compared to the other parameters. Here, $R@K^n = 0.855$ and $P@K^n = 0.0084$. These scores show that the recommendation system can identify a much larger number of goals through negative examples. This suggests that this parameter contributes to a more effective method of identification and recommendation.

Interestingly, the parameter HISTORY yields values of 0 for both $R@K^n$ and $P@K^n$, indicating that the recommendation system is unable to recommend targets in this mode correctly. A possible reason for this could be the inappropriateness of this parameter.

These results provide valuable information on the impact of RUN-MODE on the performance of the recommendation system. These observations suggest that the historical view of this data set yields a different result. A possible reason for this could be the wrong choice of negative sampling, which will be further investigated in Experiment B. Alternatively, it is possible that the number of available sessions is too small or that this parameter is inappropriate for this data set.

5.3.2. Experiment B

In Figure 5.2, the results of *Experiment B* are shown for the data set *Refurbed* with $\mathcal{K} = 10$. The experimental setup is described in subsection 5.1.2. The metrics used are defined in subsection 2.5.2. All data are available in Table A.2.

The results of runs 1-3 are provided in subsection 5.3.1, in contrast to other negative sampling methods.

Runs 1, 4, and 7 are associated with the STRUCTURAL parameter. Here, it can be

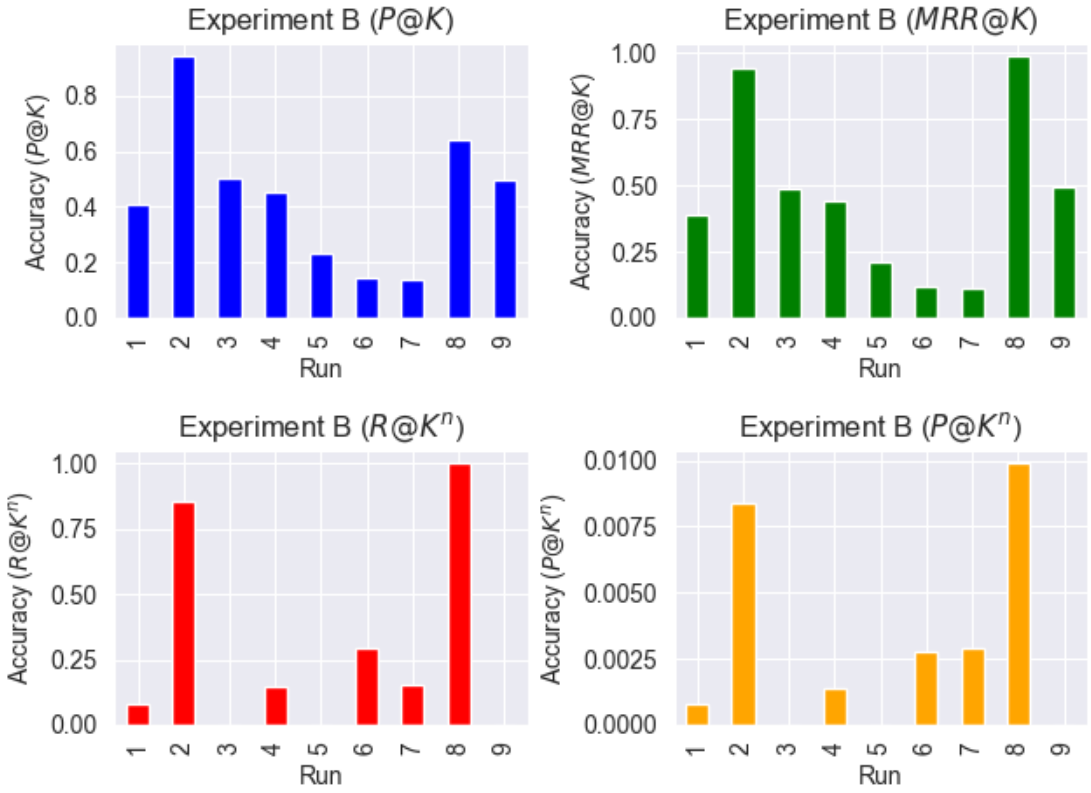


Figure 5.2.: Results of *Experiment B* for the data set *Refurbed* with $\mathcal{K} = 10$. Each bar in the graph represents a unique run of the experiment. These runs vary based on the different specifications of the hyperparameter RUN-MODE, which is used in conjunction with NEGATIVE-SAMPLER. The first, second, and third runs are associated with STRUCTURAL, SESSION, and HISTORY, respectively. In these runs, they are combined with MOST-SIMILAR. Again, the fourth, fifth, and sixth runs use STRUCTURAL, SESSION, and HISTORY. However, in these runs, they are paired with RANDOM. Lastly, the seventh, eighth, and ninth runs utilize STRUCTURAL, SESSION, and HISTORY, respectively. In these cases, they are paired with LESS-SIMILAR. This figure simplifies the analysis of the effects induced by variations in the RUN-MODE hyperparameter when combined with different NEGATIVE-SAMPLER options.

observed that the negative sampling method RANDOM performs best, yielding $P@K = 0.4548$. This result contrasts with MOST-SIMILAR, which achieves $P@K = 0.4061$, and MOST-SIMILAR, which achieves $P@K = 0.1384$. Therefore, it can be assumed that the choice of negative sampling significantly impacts this case.

The runs 2, 5, and 8 correspond to the SESSION parameter. The negative sampling method MOST-SIMILAR outperforms the others. Interestingly, RANDOM performs worse than STRUCTURAL with $P@K = 0.2298$. This finding is particularly interesting because the other two sampling methods perform better. It becomes even more intriguing when considering metrics that include negative sampling. In this experiment, 100 randomly selected negative targets are used. If randomly selected items are assumed to give inferior results for SESSION, they also give inferior results for negative items. This can be seen from the excellent results of $R@K^n = 1$ compared to $R@K^n = 0.1527$ for STRUCTURAL, where the best result is chosen for this comparison.

The HISTORY trials are described in Experiments 3, 6, and 9. It can be seen that LESS-SIMILAR gives the best results with $P@K = 0.5034$, while LESS-SIMILAR gives relatively good results with $P@K = 0.4981$. It is particularly noteworthy that the parameters HISTORY with $R@K^n = 0$ and $P@K^n = 0$ do not correctly predict any target, which was unexpected by the author. This is especially surprising given the extremely accurate results of 100% correct prediction for SESSION.

In summary, when considering only positive targets, the combination of SESSION and MOST-SIMILAR gives the best scores. However, when negative targets are also considered, LESS-SIMILAR significantly outperforms the others. Whether this changes when combined with other hyperparameters will be investigated in future experiments. Furthermore, more research is needed to determine if other hyperparameters can improve the performance of HISTORY. Considering that the running modes are built on each other, TGR-STR is considered the least suitable, as TGR-HIS has not yet been proven unsuccessful.

5.3.3. Experiment C

In Figure 5.3, the results of *Experiment C* for the data set *Refurbed* with $\mathcal{K} = 10$. The experimental setup is described in subsection 5.1.3. The metrics used are defined in subsection 2.5.2, and the results can be found in Table A.3.

The run mode is set to SESSION, and the negative sampling to MOST-SIMILAR. The impact of CATEGORY-SHIFT is examined only for TGR-SES, as it can only be considered separately in this context. The results are as follows: $P@K = 0.9854$, $P@K = 0.9847$,

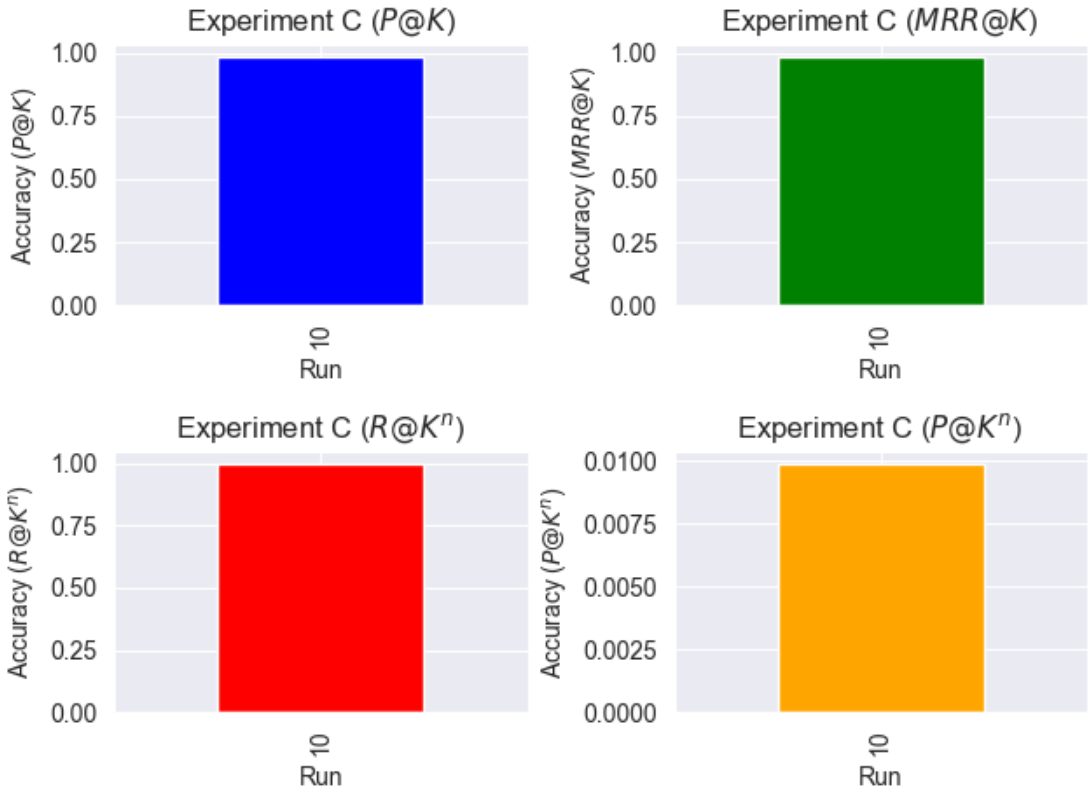


Figure 5.3.: Results of *Experiment C* for the data set *Refurbed* with $\mathcal{K} = 10$. In run 10, SESSION is applied while CATEGORY-SHIFT is not executed. The purpose is to explore the effect of this particular hyperparameter.

$R@K^n = 1$, and $P@K^n = 0.0099$. These scores indicate that this combination produces highly accurate recommendations. Especially, the influence on the negative examples is significant, as it is now 100%, similar to LESS-SIMILAR. It should be noted that the penalty for frequent CATEGORY-SHIFT was expected to have a positive influence. However, these results show that this is different. Several reasons could contribute to this result. One possibility is that the choice of penalty is inappropriate or that the data set consists mainly of purchased items, raising the question of whether people specifically search for items on such a platform. Another factor could be the need to include a threshold if the penalty is to be applied. However, these aspects were not further explored in this study and would need to be investigated in future research.

In summary, the results of this experiment highlight the high accuracy of the recommendation system in generating recommendations and its effective recommendation

ordering. In addition, it can identify positive targets while considering negative examples.

5.3.4. Experiment D

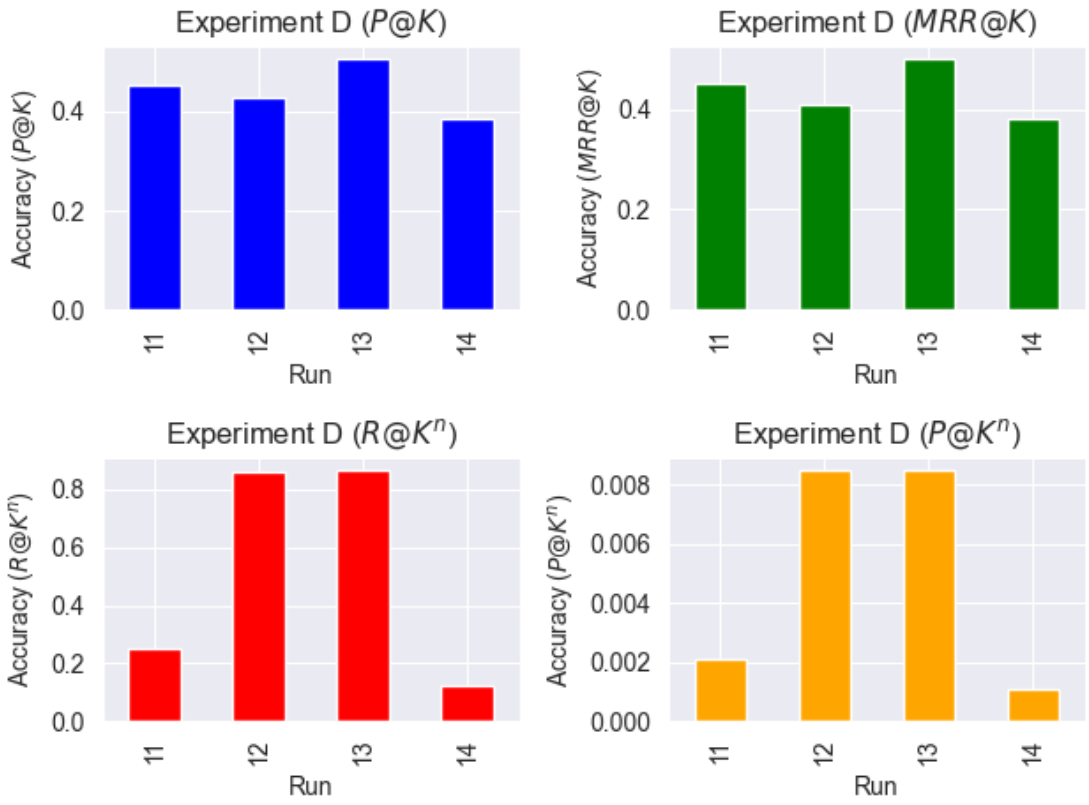


Figure 5.4.: Results of *Experiment D* for the data set *Refurbed* with $\mathcal{K} = 10$. Runs 11 through 14 apply HISTORY while excluding both CATEGORY-SHIFT and USE-SESSION-TIME-SAVE. The purpose is to investigate the effect of the USE-SESSION-TIME-SAVE hyperparameter. Specifically, runs 11, 12, and 14 do not use CATEGORY-SHIFT and USE-SESSION-TIME-SAVE. However, run 12 uniquely includes min-max normalization. In contrast, run 13 uses CATEGORY-SHIFT but excludes USE-SESSION-TIME-SAVE. Finally, run 14 uses LESS-SIMILAR, while all other runs of this experiment use MOST-SIMILAR.

In Figure 5.4, the results of *Experiment D* are shown for the data set *Refurbed* with $\mathcal{K} = 10$. The experimental setup is described in subsection 5.1.4, and the metrics used are defined in subsection 2.5.2. Result data can be found in Table A.4.

It can be observed that negative sampling has a positive effect on the metrics. In particular, $R@K^n = 0.1207$ can be observed, which, although not very satisfactory compared to the results of TGR-SES, shows a positive effect compared to $R@K^n = 0$ in previous setups. Additionally, when CATEGORY-SHIFT is disabled, and MOST-SIMILAR is used, the results are as follows $P@K = 0.4545, R@K^n = 0.1207$ in one case, and $P@K = 0.3872, R@K^n = 0.2524$ in the other. Using a min-max function instead of a sigmoid function in the score function improves the discriminability of the results, as shown in Table A.4. However, this modification does not improve the positive influence metrics, where $P@K = 0.4270$ is obtained compared to $P@K = 0.4545$. However, a significant increase is observed for $R@K^n = 0.8629$. This increase can only be explained by the fact that the randomly selected results in this test fit better than before. Otherwise, there would also be a positive effect on the other metrics. In particular, the combination of MOST-SIMILAR with CATEGORY-SHIFT enabled and USE-SESSION-TIME-SAVE disabled yields $P@K = 0.5075, R@K^n = 0.8684$.

The results highlight the significant impact of hyperparameter selection on the performance of the recommendation system. In particular, variations in USE-SESSION-TIME-SAVE resulted in significant differences in metric scores, especially compared to previous experiments on this data set. These results suggest that USE-SESSION-TIME-SAVE is a critical factor in generating recommendations. However, it remains an open question whether the inadequacy of the method lies solely in its choice or whether the entire concept of manipulating session embeddings with their timestamps to provide temporal discriminability for the attribution layer is inappropriate. Further exploration of this question is beyond the scope of this paper.

5.3.5. Experiment E

In Figure 5.5, the results of *Experiment E* for the data set *Refurbed* with $\mathcal{K} = 10$ are shown. The experimental setup is described in subsection 5.1.5. The metrics used are defined in subsection 2.5.2. All data are available in Table A.5. This experiment used the best hyperparameters used so far, namely MOST-SIMILAR and a disabled CATEGORY-SHIFT. In the first run of this experiment, the results are for the hyperparameter configuration with a minimum of 5 items, a minimum of 5 items, a maximum of 0.10 items, and a maximum of 0.10 users. The scores obtained for $P@K$ and $MRR@K$ are both 0.6079, while $R@K^n$ is 0.191 and $P@K^n$ is 0.0018.

The second run for a different hyperparameter configuration, specifically, a minimum of

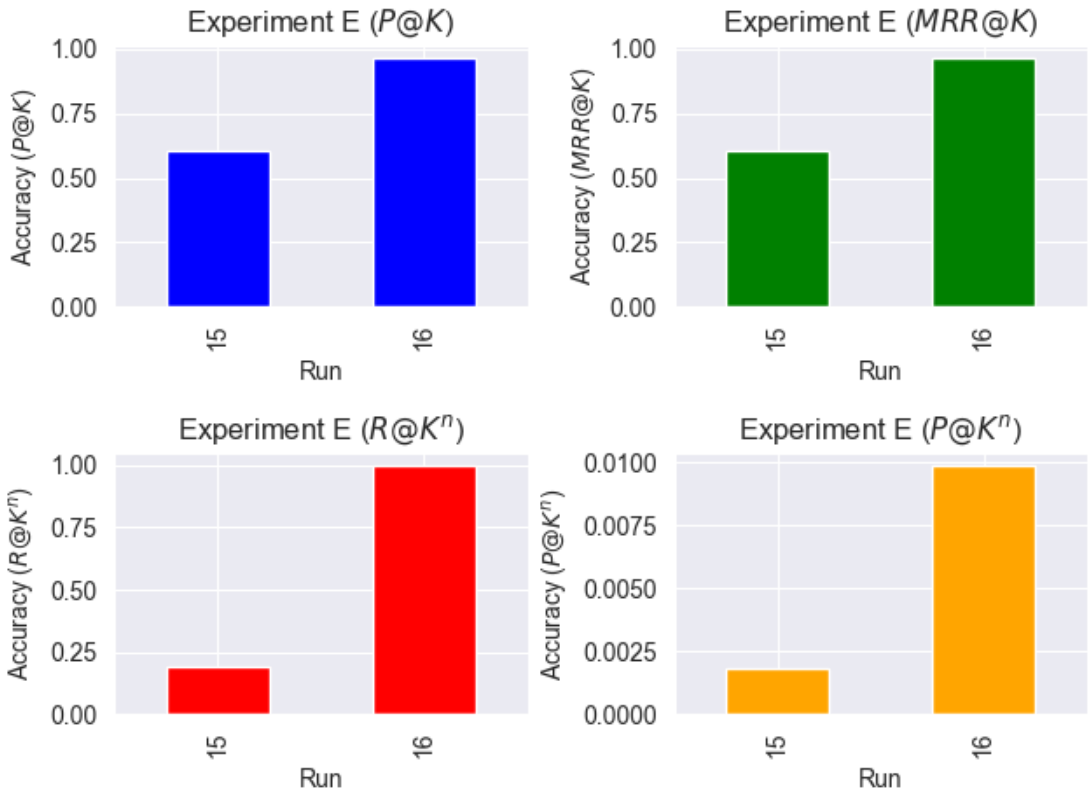


Figure 5.5.: Results of *Experiment E* for the data set *Refurbed* with $\mathcal{K} = 10$. In the graph, each bar represents a different experimental run performed with different configurations of the RUN-MODE hyperparameter. Specifically, runs 15 and 16 were run with the SESSION parameter, omitting USE-CATEGORY-SWITCH. Run 15 is designed to examine the sequence length parameter in a separate set of analyses, denoted by MIN-COUNT-USER =MIN-COUNT-ITEM =3. Conversely, run 16 examined the sequence length with the configuration REMOVE-COMMON-USERS =REMOVE-COMMON-ITEMS =1. This figure allows the study of the effects associated with different sequence lengths.

3 items, a minimum of 3 items, a maximum of 0.05 items, and a maximum of 0.05 users. In this case, the metrics $P@K$ and $MRR@K$ experienced a significant increase, reaching scores of 0.9673 and 0.9668, respectively. Furthermore, $R@K^n$ reached the maximum value of 1, while $P@K^n$ showed a value of 0.0099.

The analysis of the results highlights the significant impact of varying hyperparameters on the performance of the recommendation systems. In particular, reducing the minimum number of items and items while increasing the maximum number of items and users led to a significant improvement in the metrics $P@K$ and $MRR@K$. Additionally, it should be noted that the second hyperparameter configuration showed a significantly higher detection rate for negative examples, as evidenced by the value $R@K^n$ of 1.

Based on these results, the configuration with a minimum of 3 items, a minimum of 3 items, a maximum of 0.05 items, and a maximum of 0.05 users is the most suitable for recommendation systems in this scenario. This observation is particularly interesting because there is little deviation from the minimum number of 5, indicating that even short sessions can be considered. This can serve as a starting point for further research. The question of determining appropriate thresholds for the two parameters remains open. However, this experiment clearly shows that the choice of minimum and maximum numbers significantly impacts the results.

5.3.6. Conclusion

This subsection summarizes the experiments performed on the *Refurbed* data set. The Figure 5.6 shows the results of all experiments for the *Refurbed* data set with $\mathcal{K} = 10$. The metrics used for the evaluation are defined in subsection 2.5.2.

Experiment A focused on the *Refurbed* data set and showed that using the SESSION mode in conjunction with the MOST-SIMILAR negative sampling yielded the best results. TGR achieved an impressive $P@K$ of 0.9429 and an almost identical $MRR@K$ of 0.9420. On the contrary, the other two parameters for RUN-MODE, namely STRUCTURAL and HISTORY, achieved lower performance values. It should be noted that $P@K$ and $MRR@K$ were closely aligned in all experiments, which may indicate a potential problem within the network architecture. However, this requires further investigation beyond the scope of this paper.

Experiment B compared different negative sampling schemes in all three RUN-MODE parameters. The results showed that combining MOST-SIMILAR with SESSION produced the best results while combining LESS-SIMILAR with HISTORY produced optimal results.

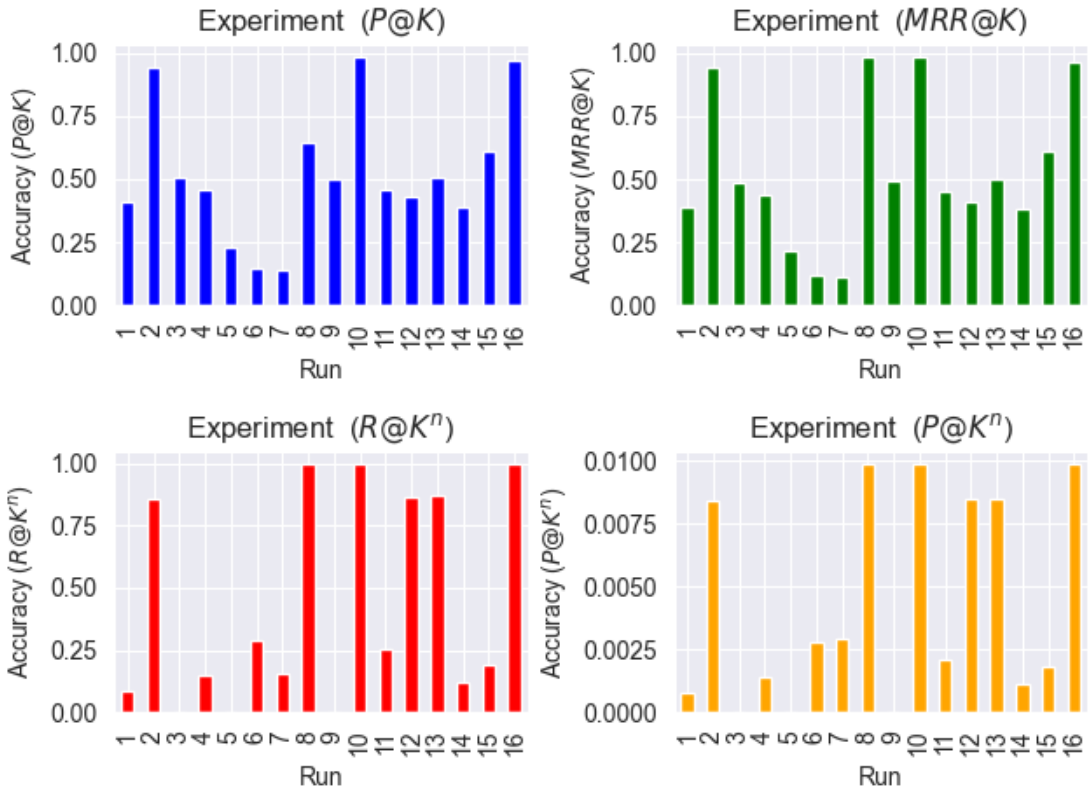


Figure 5.6.: Results of the experiments for the data set *Refurbed* with $\mathcal{K} = 10$

This highlights the significant impact of negative sampling on the metrics.

Experiment C explored the impact of CATEGORY-SHIFT on SESSION and concluded that it had no positive effect. The recommendation system showed high accuracy and good recommendation ordering.

Experiment D investigated the influence of different hyperparameters on the performance of the recommendation system. The results showed that the choice of the USE-SESSION-TIME-SAVE hyperparameter resulted in significant variations in the metric values.

Experiment E used the best hyperparameter configurations and showed the significant impact of hyperparameter selection affecting per-user elements on the performance of the recommendation system.

In summary, the SESSION mode combined with MOST-SIMILAR for negative sampling showed the most significant impact on the performance of the recommendation system. On the contrary, the change in the category had no positive effect. The choice of

hyperparameters, especially negative sampling and the parameter USE-SESSION-TIME-SAVE, also played an important role in shaping the metric values.

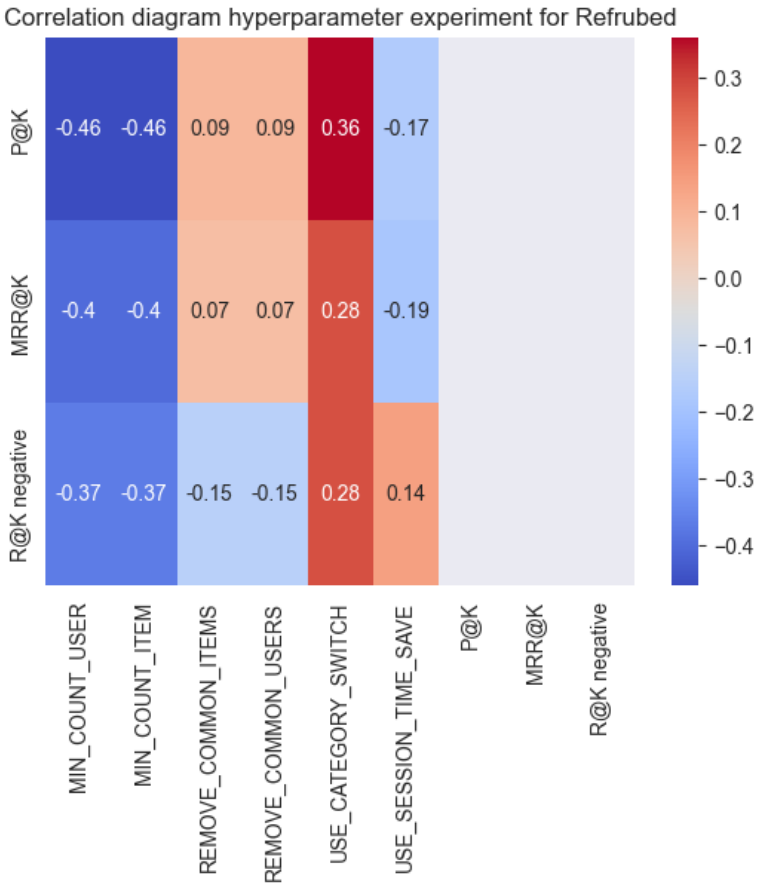


Figure 5.7.: Correlation diagram of the hyperparameters for the data set *Refurbed* with $\mathcal{K} = 10$

A correlation matrix is shown in Figure 5.7 to explore the interdependencies among the hyperparameters. The correlation coefficients in the matrix provide insight into the relationships between the metrics. A correlation coefficient of 1 indicates a perfect positive correlation, while -1 indicates a perfect negative correlation. Values close to 0 indicate weak or no correlation. An interesting observation is that MIN-COUNT-USER or MIN-COUNT-ITEM have a relatively strong positive correlation. However, this correlation does not hold for REMOVE-COMMON-USERS or REMOVE-COMMON-ITEMS. It is necessary to emphasize that since these parameters were only examined in a single experiment, it is difficult to draw definitive conclusions about their correlations. For hyperparameters

Model	Criterion	Min Nodes	Max Nodes	$R@K^n$
HSAGEConvGAE	BPRLoss	5	0.05	0.864
HSAGEConvGAE	BPRLoss	5	0.1	0.9043
HSAGEConvGAE	BPRLoss	10	0.1	0.9259
HSAGEConvGAE	MarginRankingLoss	10	0.1	0.9347
HUniGINConvGAE	BPRLoss	5	0.05	0.804
TGR (Experiment: 8)	BCELoss	5	0.05	1
TGR (Experiment: 10)	BCELoss	5	0.05	1
TGR (Experiment: 16)	BCELoss	3	0.05	1

Table 5.6.: Results of the experiments for the data set *Refurbed* (Other works based on UniGNN). Source: [Kum23]

USE-CATEGORY-SWITCH and USE-SESSION-TIME-SAVE, a moderate positive and negative correlation was observed, respectively. This suggests that the assumption made in the experiment of a strong influence of these parameters on the data set may not be true. It is crucial to acknowledge that this analysis is based on the available data, and further investigation and verification are necessary to gain a deeper understanding of the relationships between the metrics. Furthermore, it is important to remember that correlations alone do not imply causality. Further experimental investigation and analysis may be required to validate the metrics' underlying relationships.

5.3.7. Comparison with other works

The results presented in Table 5.6 are based on internal research at the University of Vienna based on UniGNN (Unified Framework for Graph and Hypergraph Neural Networks). This framework unifies graph and hypergraph modeling and learning with NNs. UniGNN extends the concept of GNN and allows integration of hypergraphs into the learning process to solve the problem that graphs and hypergraphs have traditionally been considered separately due to their different structural properties [HY21]. The modules shown here are also based on having a minimum number of nodes and a maximum frequency. The *Min Nodes* and *Max Nodes* shown in the table stand for MIN-COUNT-USER = MIN-COUNT-ITEM = *Min Nodes* and REMOVE-COMMON-USERS = REMOVE-COMMON-ITEMS = *Max Nodes*.

As seen in Table 5.6, TGR performs better, for example, in run 10. However, it should

be noted that this is only a comparison $R@K^n$, as a reliable comparison between the models $P@K$ and $MRR@K$ would also be needed. It is also true that the data presented here from the University of Vienna are preliminary, so a full comparison would have to be done in future work.

5.3.8. Open topics

This section presents a list of research challenges based on the findings of the experiments in the data set *Refurbed*.

$MRR@K - P@K$ The observations indicate that when an item is hit in $P@K$, it is unlikely to have a position of 1. A similar trend is expected for the corresponding values of $MRR@K$ [Wu+20a; Gao+21]. These results may indicate possible inconsistencies in the model that require further investigation.

Criterion Alternative criteria that may better fit this model have not yet been evaluated. Research in this direction would be beneficial.

use-session-time-save Further empirical research is warranted to assess the suitability of this hyperparameter. In addition, possible alternative formulations for optimizing its value should be explored.

use-category-switch A comprehensive analysis is required to evaluate the suitability of this hyperparameter. The exploration of other methods such as KG can also be beneficial.

Min/Max number of edges More research is needed to determine how many edges each node (user or item) requires in the bipartite graph so that this model does not lose its ability to generate recommendations. Similarly, more research is needed to determine if a maximum number of edges reduces the predictive quality, as the prediction of these is more likely than more unknown nodes.

In general, there are still uncertainties regarding the suitability of this model and potential avenues for improvement. Determining the threshold at which the model can no longer effectively detect and predict sessions requires further investigation. Thus, it currently needs to be more conclusive whether this model is appropriate for the *Refurbed* data set.

5.4. Retailrocket

This section describes the experiments and their results obtained with the *RetailRocket* dataset, available in section 4.4.

5.4.1. Experiment A

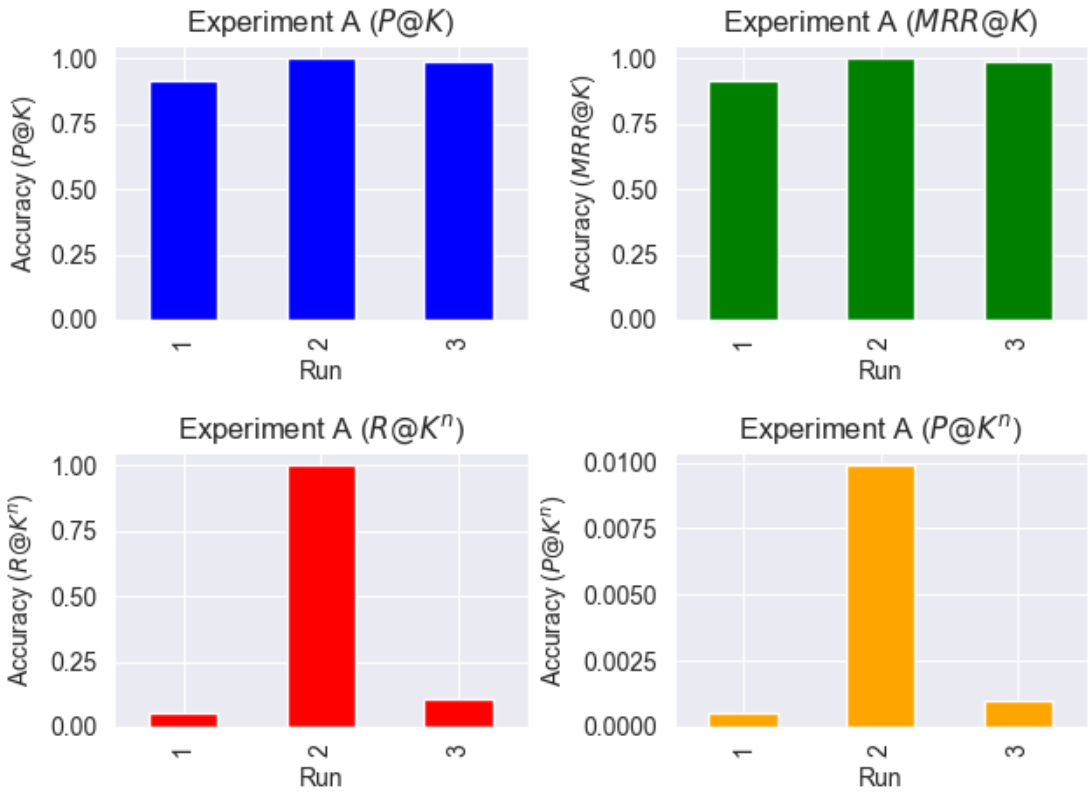


Figure 5.8.: Results of *Experiment A* for the data set *RetailRocket* with $\mathcal{K} = 10$. Each bar represents a run of the experiment performed under different specifications of the hyperparameter RUN-MODE. The first, second, and third experimental runs are associated with STRUCTURAL, SESSION, and HISTORY, respectively. This graphical representation facilitates an analytical inspection of the effects induced by variations in the RUN-MODE hyperparameter.

In Figure 5.8, the results of *Experiment A* are shown for the data set *RetailRocket* with $\mathcal{K} = 10$. The experimental setup is described in subsection 5.1.1. The metrics used are defined in subsection 2.5.2. The complete result is available at Table A.11.

Using STRUCTURAL, the recommendation system achieves a high accuracy of 0.9115 for the metric $P@K$ and a corresponding score of 0.9115 for the metric $MRR@K$. These results indicate that the system can accurately predict the top $\mathcal{K}$ recommendations. However, the value of $R@K^n$ is relatively low at 0.0505, suggesting that the system has difficulties excluding irrelevant items based on negative sampling. Furthermore, the precision for negative items is only 0.0005, suggesting a potential lack of reliability in identifying irrelevant recommendations.

In the case of SESSION, the system achieves $P@K = MRR@K = 1$. This performance signifies the system's ability to accurately identify the correct recommendations for each user and their current session. The metrics that consider negative sampling also achieve exceptional scores of $R@K^n = 1$, indicating that the system can successfully exclude all irrelevant items.

In the context of SESSION, a precision of $P@K = MRR@K = 0.9857$ is observed, indicating a high level of accuracy in the integration of the user's history. However, the value of $R@K^n$ is only 0.1082, indicating that the system has difficulty excluding irrelevant items. Therefore, there is potential for improvement.

In general, it can be concluded that SESSION achieves the best results across all evaluation metrics, demonstrating the ability of the recommendation system to generate accurate recommendations for individual user sessions. The other two parameters of RUN-MODE show difficulties in excluding negative sample items.

These results generally confirm the expected results and emphasize the importance of the selected hyperparameters for the performance of GNN-based recommendation systems. However, more research is needed. In particular, exploring why the TGR-HIS results show unexpectedly weak performance when negative items are considered is interesting. Future experiments in this thesis will shed light on whether alternative hyperparameters can improve this aspect.

5.4.2. Experiment B

In Figure 5.9, the results of *Experiment B* are shown for the data set *RetailRocket* with $\mathcal{K} = 10$. The experimental setup is described in subsection 5.1.2. The metrics used are defined in subsection 2.5.2. All results are available in Table A.12. The results of runs 1-3 are provided in subsection 5.4.1 and compared with the other negative sampling methods in this section.

Runs 1, 4, and 7 investigated the STRUCTURAL parameter. When using the negative

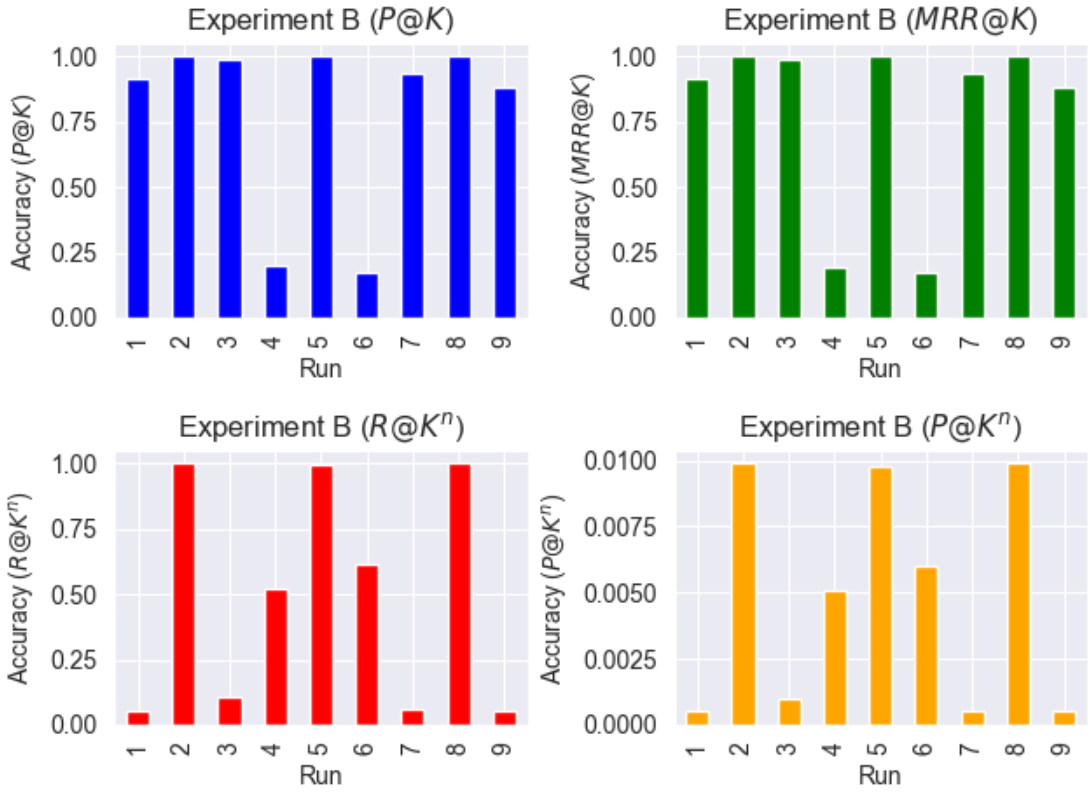


Figure 5.9.: Results of *Experiment B* for the data set *RetailRocket* with $\mathcal{K} = 10$. Each bar in the graph represents a unique run of the experiment. These runs vary based on the different specifications of the hyperparameter RUN-MODE, which is used in conjunction with NEGATIVE-SAMPLER. The first, second, and third runs are associated with STRUCTURAL, SESSION, and HISTORY, respectively. In these runs, they are combined with MOST-SIMILAR. The fourth, fifth, and sixth runs again use STRUCTURAL, SESSION, and HISTORY. However, in these runs, they are paired with RANDOM. Lastly, the seventh, eighth, and ninth runs utilize STRUCTURAL, SESSION, and HISTORY, respectively. In these cases, they are paired with LESS-SIMILAR. This figure simplifies the analysis of the effects induced by variations in the RUN-MODE hyperparameter when combined with different NEGATIVE-SAMPLER options.

sampling method RANDOM, the lowest scores were $P@K = 0.1974$. On the other hand, when using LESS-SIMILAR, the results were much better with $P@K = 0.9358$. It is important to emphasize that LESS-SIMILAR is not the optimal sampling method. Interestingly, the score $R@K^n$ did not reach a satisfactory level for any of the negative sampling methods, indicating that this aspect of the recommendation system is unstable regardless of the negative sampling method used. However, Interestingly, the best $R@K^n$ value of 0.5221 was obtained with RANDOM.

Runs 2, 5, and 8 focus on SESSION. The results of this analysis indicate that the choice of the negative sampling method does not significantly impact this run mode. Only a small negative impact of 0.0001 was observed for $P@K$ and 0.0037 for $R@K^n$ when using RANDOM. These results suggest that the model consistently rejects negative items and effectively incorporates positive and negative user preferences.

Experiments on HISTORY are described in runs 3, 6, and 9. Among these runs, MOST-SIMILAR showed the best results with $P@K = 0.9857$, while LESS-SIMILAR also performed well with $P@K = 0.8805$. Interestingly, RANDOM performed less well than the other two runs, which was unexpected. However, it still produced the best $R@K^n$ value of 0.6118 for this run mode. In particular, similar to STRUCTURAL, the combination of HISTORY and the negative sampling methods resulted in the best performance in the negative metrics domain.

In summary, the model SESSION showed the best overall performance in all metrics and demonstrated an excellent ability to generate relevant recommendations.

5.4.3. Experiment C

In this section, the results of *Experiment C* are presented, which was performed on the *RetailRocket* data set with $\mathcal{K} = 10$, as shown in Figure 5.10. The experimental setup is described in detail in subsection 5.1.3. The evaluation metrics used in this study are defined in subsection 2.5.2. The complete results are in Table A.13.

For this experiment, the run mode was set to SESSION, and the negative sample was set to MOST-SIMILAR. The effect of category changes was investigated only for TGR-SES since it can be considered independently in this particular case. The results are as follows: $P@K = MRR@K = R@K^n = 1$. These values show that this combination does not significantly degrade the precision of the recommendation. The results are consistent with those of the previous experiments.

In general, the results of this experiment indicate that the recommendation system

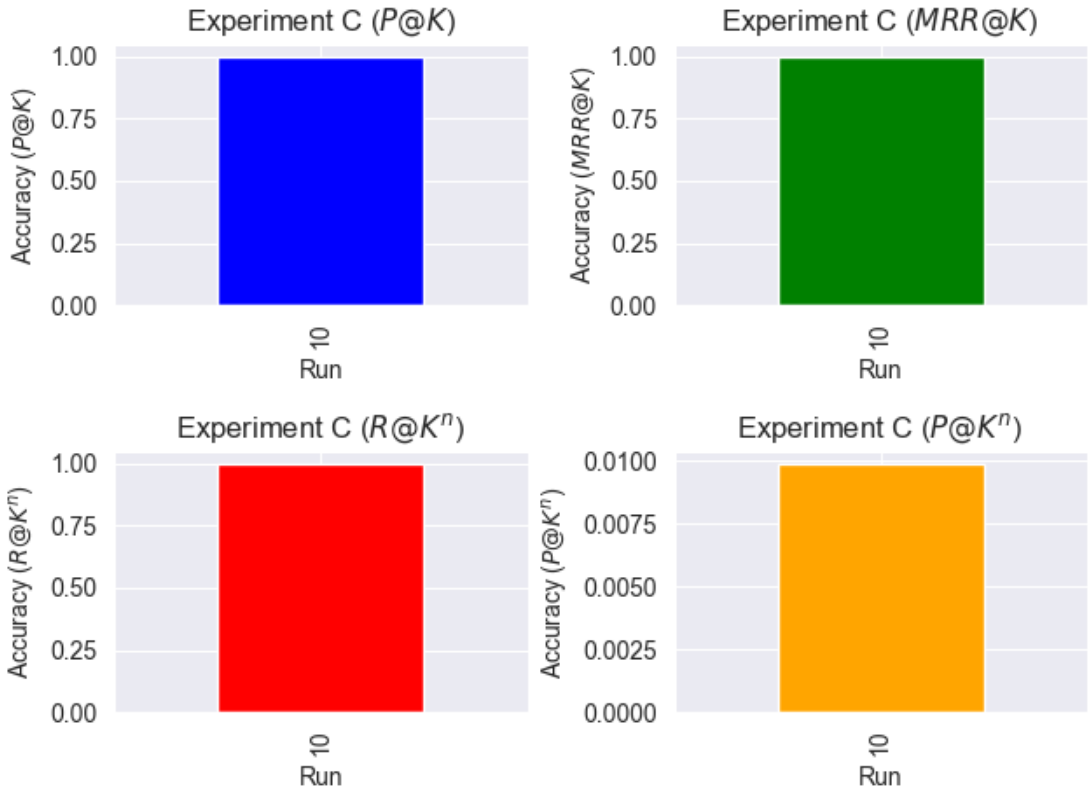


Figure 5.10.: Results of *Experiment C* for the data set *RetailRocket* with $\mathcal{K} = 10$. In run 10, SESSION is applied while CATEGORY-SHIFT is not executed. The purpose is to explore the effect of this particular hyperparameter.

achieves a high level of accuracy in generating recommendations.

5.4.4. Experiment D

In Figure 5.11, the results of *Experiment D* for the data set *RetailRocket* with $\mathcal{K} = 10$ are shown. The experimental setup is described in subsection 5.1.4. The metrics used are defined in subsection 2.5.2. All results are available in Table A.14.

The first combination is MOST-SIMILAR without using CATEGORY-SHIFT and USE-SESSION-TIME-SAVE. In this setting, the model achieved values of $P@K = MRR@K = 1$, indicating that it correctly identified all relevant elements of the recommendation. The model prioritized the positive target over the negative examples, demonstrating its ability to capture the complexity of the data and effectively model the differences between

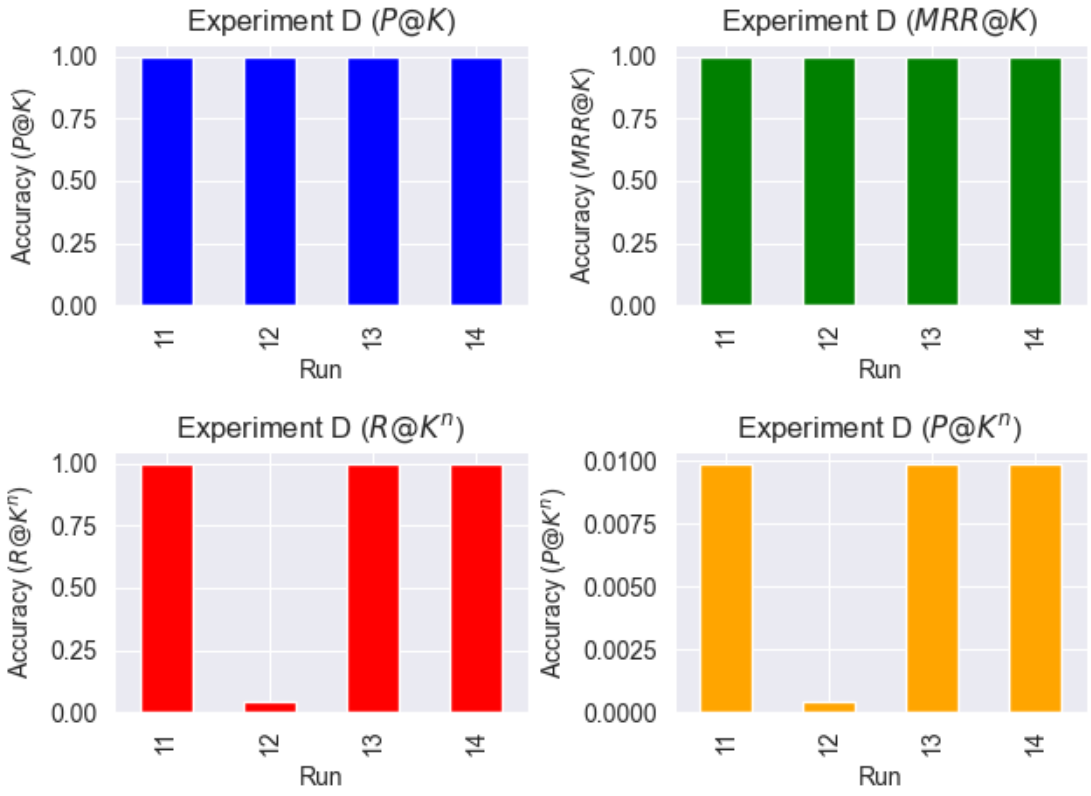


Figure 5.11.: Results of *Experiment D* for the data set *RetailRocket* with $\mathcal{K} = 10$. Runs 11 through 14 apply HISTORY while excluding both CATEGORY-SHIFT and USE-SESSION-TIME-SAVE. The purpose is to investigate the effect of the USE-SESSION-TIME-SAVE hyperparameter. Specifically, runs 11, 12, and 14 do not use CATEGORY-SHIFT and USE-SESSION-TIME-SAVE. However, run 12 uniquely includes min-max normalization. In contrast, run 13 uses CATEGORY-SHIFT but excludes USE-SESSION-TIME-SAVE. Finally, run 14 uses LESS-SIMILAR, while all other runs of this experiment use MOST-SIMILAR.

positive and negative examples. As a result, the score $R@K^n$ is 1.

Run 12 is similar to run 11, adding a Min-Max function instead of a sigmoid function in the scoring function to evaluate the discriminability of the results scores in Table A.14. Although a slightly larger score difference was achieved, the metrics were negatively affected. The range $R@K^n$ decreased significantly to 0.0431, indicating that the model struggled to score negative elements lower and exclude them from the recommendations. Furthermore, a low positive score of $P@K = MRR@K = 0.995$ was obtained, which is

still considered good.

The last two combinations alternate between USE-CATEGORY-SWITCH and USE-SESSION-TIME-SAVE. The results are identical to those of run 11.

These results only partially illustrate the significant influence of hyperparameters on TGR. In particular, using USE-SESSION-TIME-SAVE or USE-CATEGORY-SWITCH leads to noticeable variations in the metric values, especially compared to previous experiments conducted on this data set using TGR-HIS. These results suggest that USE-SESSION-TIME-SAVE or USE-CATEGORY-SWITCH are critical in the generation of recommendations. However, the exact impact of these hyperparameters on this data set remains unclear. One seems required, but it does not make a difference whether one or both parameters are used as long as one is enabled. This behavior can only be explained by the fact that both parameters improve the model enough to allow accurate predictions, but further investigation is needed in future research.

5.4.5. Experiment E

In Figure 5.24, the results of *Experiment E* for the data set *Diginetica* with $\mathcal{K} = 10$ are presented. The experimental setup is described in detail in subsection 5.1.5. The evaluation metrics used in this experiment are defined in subsection 2.5.2. As a table, the results can be found in Table A.10.

This experiment uses the most effective hyperparameters based on previous runs for TGR-SES and TGR-HIS. Specifically, runs 15 and 16 used TGR-SES with MOST-SIMILAR and CATEGORY-SHIFT disabled. On the other hand, runs 17 and 18 used TGR-HIS with MOST-SIMILAR disabled and the USE-SESSION-TIME-SAVE parameter excluded, while USE-CATEGORY-SWITCH was enabled for run 18 and disabled for run 17. These runs aimed to investigate whether TGR-SES performs better with different numbers of items per user and whether this dataset's previously determined optimal configuration remains equally effective with different edges per user node.

Interestingly, all runs consistently demonstrate perfect predictive quality, as indicated by $P@K = MRR@K = R@K^n = 1$. This outcome was unexpected to the author, as it would be reasonable to assume that a change in sequence length would have a substantial impact. However, this model was trained with a minimum sequence length of 2, which could explain these results.

Based on these findings, the optimal configuration for the recommendation systems cannot be determined, as it appears to have no discernible influence. This result is

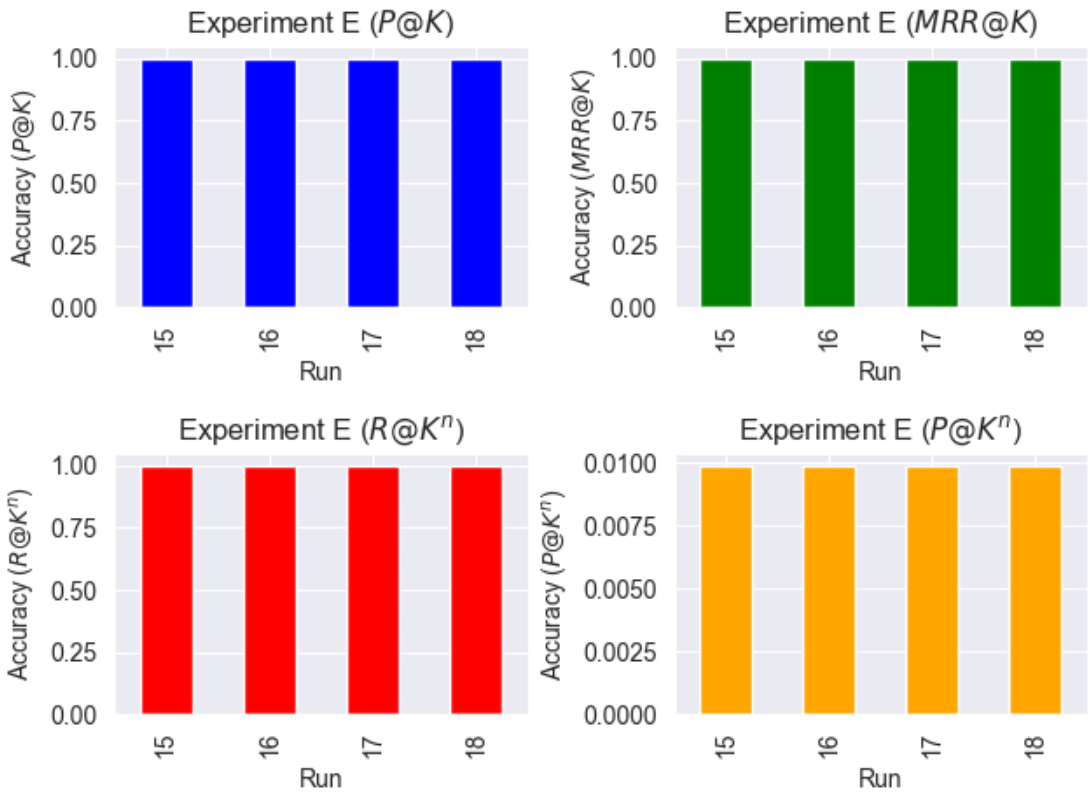


Figure 5.12.: Results of *Experiment E* for the data set *RetailRocket* with $\mathcal{K} = 10$. The graph illustrates various experimental runs. Runs 15 and 16 utilized the SESSION parameter, excluding USE-CATEGORY-SWITCH, while runs 17 and 18 employed the HISTORY parameter, excluding both USE-CATEGORY-SWITCH and USE-SESSION-TIME-SAVE. Additionally, runs 15 and 17 were designed to study the sequence length parameter, represented by MIN-COUNT-USER =MIN-COUNT-ITEM =3. In contrast, runs 16 and 18 explored sequence length with the configuration REMOVE-COMMON-USERS =REMOVE-COMMON-ITEMS =1. Default hyperparameter settings were applied in scenarios not explicitly mentioned.

particularly remarkable since varying the minimum number yields no significant changes, suggesting that even short sessions can be considered. These results serve as a valuable starting point for further investigation. Although the question of setting appropriate thresholds for these two parameters remains unanswered, this experiment offers insight, as the choice of MIN-COUNT-USER, MIN-COUNT-ITEM and REMOVE-COMMON-USERS,REMOVE-COMMON-ITEMS significantly affects the results.

5.4.6. Conclusion

This section summarizes the experiments conducted on the *RetailRocket* data set. The Figure 5.13 shows the results of all experiments conducted on the data set *RetailRocket* with $\mathcal{K} = 10$. The evaluation metrics used are defined in the background subsection 2.5.2.

Experiment A examined the effect of hierarchical architecture on network prediction accuracy. *Experiment A* showed that using the SESSION mode in combination with MOST-SIMILAR produced the best results. The recommendation system achieved an impressive $P@K$ value of 1, corresponding to a $MRR@K$ value of 1. On the contrary, the other two parameters of RUN-MODE, namely STRUCTURAL and HISTORY, showed lower performance values for the $R@K^n$ metric. Interestingly, the values for $P@K$ and $MRR@K$ were consistently close in all experiments, possibly indicating a network problem. However, a more detailed investigation of this issue is beyond the scope of this paper.

Experiment B compared different negative sampling methods for the three RUN-MODE parameters. The results showed that the combination of MOST-SIMILAR with SESSION produced the best results, while the combination with RANDOM significantly degraded the results. Therefore, the choice of negative sampling significantly impacted evaluation metrics.

Experiment C examined the effect of CATEGORY-SHIFT on SESSION and found no positive effect. The accuracy of the recommendation system remained unchanged.

Experiment D investigated the influence of different hyperparameters on the performance of the recommendation system. The experiment showed that negative sampling positively impacted the evaluation metrics, and the choice of hyperparameter USE-SESSION-TIME-SAVE led to significant variations in the metric values.

Experiment E used optimal hyperparameter configurations and highlighted the significant effects of choosing the MIN-COUNT-USER, MIN-COUNT-ITEM, and REMOVE-COMMON-USERS,REMOVE-COMMON-ITEMS of interactions per node on the performance of the recommendation system. No performance changes were observed based on the metrics

used.

The *RetailRocket* data set experiments showed that SESSION and HISTORY provided the most accurate recommendations. They achieved high values for the metrics P@K, $MRR@K$, and $R@K^n$, demonstrating the accurate identification of relevant recommendations. STRUCTURAL had difficulty excluding irrelevant elements. The choice of hyperparameters, especially negative sampling in the run mode, also significantly impacted the metric values.

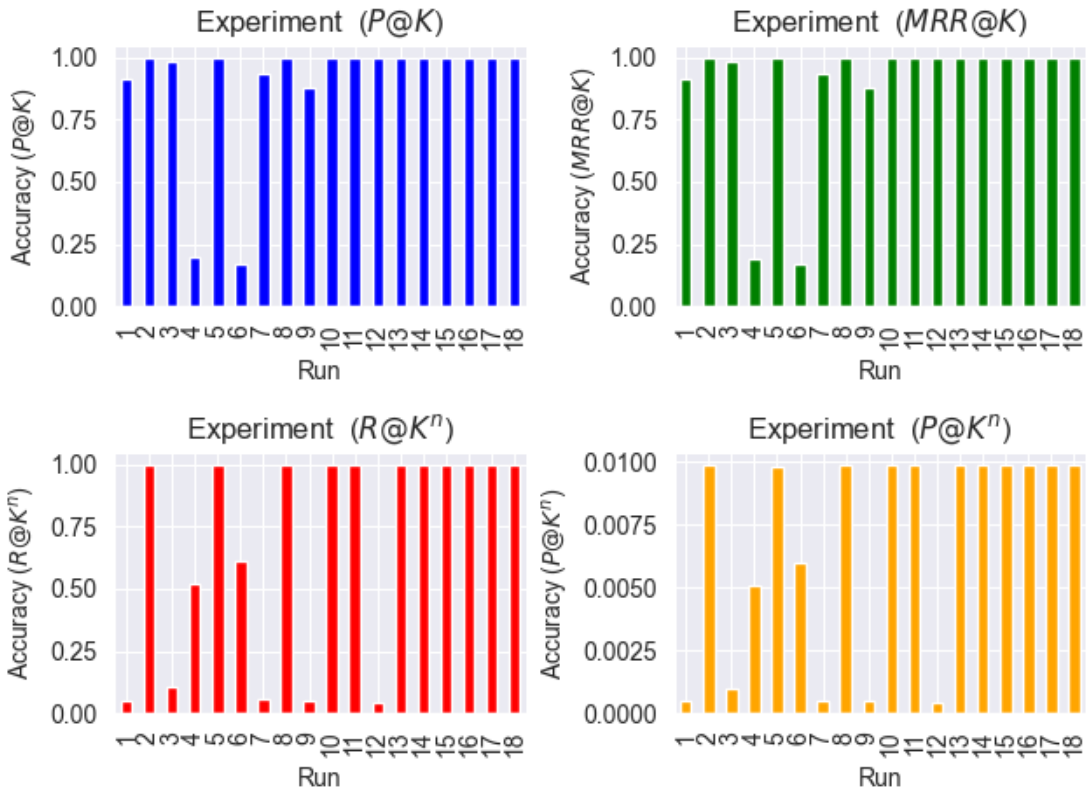


Figure 5.13.: Results of the experiments for the data set *RetailRocket* with $\mathcal{K} = 10$

A correlation matrix measures the interdependencies between the hyperparameters, as shown in Figure 5.14. The matrix's correlation coefficients indicate the associations' strength and direction. A coefficient of 1 indicates a perfect positive correlation, while -1 indicates a perfect negative correlation. Values close to 0 indicate minimal or negligible correlation.

It is significant that the metrics and MIN-COUNT-USER or MIN-COUNT-ITEM do not show a strong positive correlation, unlike REMOVE-COMMON-USERS or REMOVE-COMMON-

ITEMS. However, since these parameters were only examined in two experiments, it is impossible to determine the extent of these correlations conclusively.

Regarding the hyperparameter USE-CATEGORY-SWITCH, a moderate positive correlation is observed. This means that the experiment results indicate a significant influence of USE-CATEGORY-SWITCH, which means that it should not be used. Similarly, a weaker positive correlation is observed for USE-SESSION-TIME-SAVE.

It is important to note that this analysis is based on the available data, and further research and verification are needed to understand the relationships between the metrics better. It is also important to note that correlations alone do not establish causality. Additional experimental research and analysis are required to confirm the underlying relationships of the measured variables. However, the number of minimum elements per node has the greatest influence, consistent with findings from other studies. The feasibility and effectiveness of the various strategies remain controversial and should be further explored in future research.

5.4.7. Comparison with other works

The Table 5.9 presents the results of different models to evaluate the recommendation systems using metrics $P@20$ and $MRR@20$. Models are compared with different runs of the TGR model, which achieved different performance values in experiments. The results are generally taken from Hou et al. [Hou+22] to ensure consistency in interpreting the results, referencing individual models in the appropriate sections. Descriptions of the experiments can be found in the appropriate subsections of this section or, more generally, in section 5.1.

Run 20 is similar to run 14, except that no minimum or maximum filtering of nodes is applied to ensure a fair comparison with the other models.

The models compared include *FPMC*, *GRU4Rec*, *NARM*, *SR-GNN*, *NISER+*, *LESSR*, *SGNN-HN*, *SASRec*, *GC-SAN*, *CL4Rec*, and *CORE*. These models have received considerable attention in previous studies, and their performance for various scenarios and data sets has been extensively investigated.

The table demonstrates that the TGR consistently achieves impeccable values for $P@20$ and $MRR@20$, regardless of whether the minimum number of interactions per node is reduced, outperforming other GNN-based models. For example, in run 17, the $P@20$ value is 1, in contrast to the $P@20 = 52.89$ obtained for *CORE*. This trend remains consistent even in run 20, where a $P@20 = 1$ is recorded.

Model	Reference	P@20	MRR@20
FPMC	section 5.2	46.04	21.95
GRU4Rec	section 5.2	55.32	33.18
NARM	section 5.2	58.65	34.69
SR-GNN	section 5.2	60.36	37.43
NISER+	section 5.2	56.22	37.11
LESSR	section 5.2	58.82	35.72
SGNN-HN	section 5.2	58.82	35.72
SASRec	section 5.2	59.81	36.03
GC-SAN	section 5.2	59.69	35.95
CL4Rec	section 5.2	59.69	35.95
CORE	section 5.2	61.85	38.76
TGR (Experiment: 3)	0.98	0.98	
TGR (Experiment: 14)	1	1	
TGR (Experiment: 17)	1	1	
TGR (Experiment: 20)	1	1	

Table 5.7.: Results of the experiments for the data set *RetailRocket*, Results based on: [Hou+22]

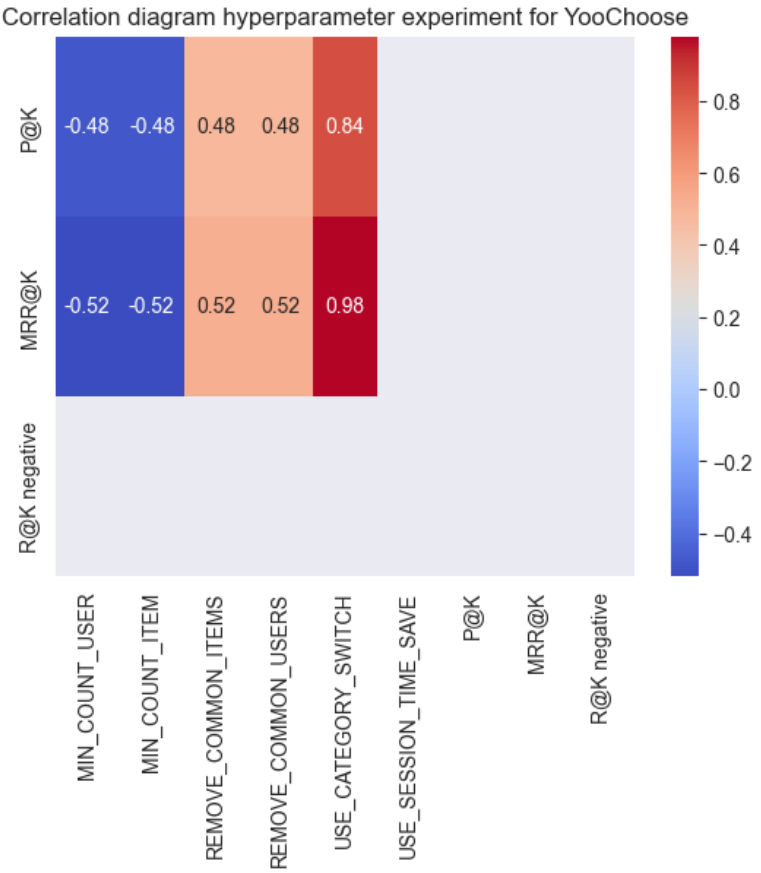


Figure 5.14.: Correlation diagram of the hyperparameters for the data set *RetailRocket* with $\mathcal{K} = 10$

These data suggest that the TGR model matches or exceeds the accuracy and recommendation classification performance of established GNN-based models when no minimum number of nodes is used. However, a direct comparison between these models is often not feasible, as TGR was explicitly designed for internal comparison with the *Refurbed* data set and is therefore not optimized for small session sizes, a feature that has been touted as beneficial in other studies [Wu+20a]. Despite these limitations, a score of $P@20 = 1$ was still obtained, a surprising and encouraging result.

Future research should confirm the reproducibility of these findings and examine the possible impact of δ between $P@K$ and $MMR@K$. The author suspects that this discrepancy may introduce bias in the results, and further investigation is warranted.

In summary, the analysis shows that the TGR model has effectively addressed the

problem of the *RetailRocket* dataset, outperforming traditional GNN-based models regarding precision and classification accuracy. However, the relatively small δ between $P@K$ and $MMR@K$ requires further investigation before concluding. It is important to note that this comparison focuses only on positive metrics. An analysis that includes the metric $R@K^n$ would be beneficial, as it could reveal potential advantages of the TGR model or expose the strengths and weaknesses of competing models.

5.4.8. Open topics

This section presents a list of research topics that should be addressed in future research based on the *RetailRocket* data set.

$MRR@K - P@K$ The observations indicate that when an item is hit in $P@K$, it is unlikely to have a position of 1. A similar trend is expected for the corresponding values of $MRR@K$ [Wu+20a; Gao+21]. These results may indicate possible inconsistencies in the model that require further investigation.

Criterion Alternative criteria that may better fit this model have not yet been evaluated. Research in this direction would be beneficial.

Min/Max number of edges For this model to not lose its ability to generate recommendations, more research is needed to determine how many edges each node (user or item) requires in the bipartite graph. Similarly, more research is needed to determine whether a maximum number of edges reduces prediction quality, since prediction of these edges is more likely than that of more unknown nodes.

use-session-time-save or use-category-switch It is unclear whether both parameters provide similar improvements and which is most appropriate for this data set. More research is needed in this regard. Furthermore, the functions behind these methods could be examined to see if similar approaches are more appropriate.

embedding-dimension-for-user for use-dimension-for-user The optimal hidden size for this data set has not been adequately investigated. More research is needed to determine the most appropriate hidden size of this particular task.

In general, uncertainties remain regarding the suitability of this model and possible avenues for improvement. Thorough research is needed to determine the threshold beyond which the model's ability to detect and predict sessions diminishes. Overall, it can be

declared that TGR is suitable for the *RetailRocket* data set and gives perfect results in certain situations with a minimum number of items in a session.

5.5. YooChoose

This section describes the experiments and their results obtained with the *YooChoose* dataset, available in section 4.3. Due to the faster network training time, *Yoochoose 1/4* is chosen, as described in subsection 2.5.1. The hierarchical architecture confirmed in section 5.5 is used.

5.5.1. Experiment A

In Figure 5.15, the results of *Experiment A* are shown for the data set *YooChoose* with $\mathcal{K} = 10$. The experimental setup is described in subsection 5.1.1. In contrast to other datasets, the parameter RANDOM applies to NEGATIVE-SAMPLER, which is explained in subsection 5.5.2. The metrics used are defined in subsection 2.5.2. All data are available in Table A.16.

The results demonstrate that RUN-MODE when paired with the parameter SESSION and coupled with negative sampling through RANDOM, provides the best results. The recommendation system achieves an excellent $P@K$ value of 0.9998, closely followed by a $MRR@K$ value of 0.9996. These elevated values suggest that the system can consistently generate a substantial fraction of correct recommendations within the top $\mathcal{K}$ recommendations, and the position of these recommendations is nearly perfect.

In contrast, the other two parameters for RUN-MODE, namely STRUCTURAL and HISTORY, yield similar results. The STRUCTURAL produces a $P@K$ value of 0.9996, while the HISTORY produces a $P@K$ value of 0.9998.

Such similarities extend to metrics $R@K^n = 1$ and $P@K^n$ with a value of 0.0099. These numbers imply that the recommendation system can identify and recommend the target item based on negative examples.

These results provide valuable information on the influence of RUN-MODE on the performance of the recommendation system. As observed, SESSION and HISTORY yield equivalent values, as described in subsection 5.5.4. The reason for this is that there are no sequential user data.

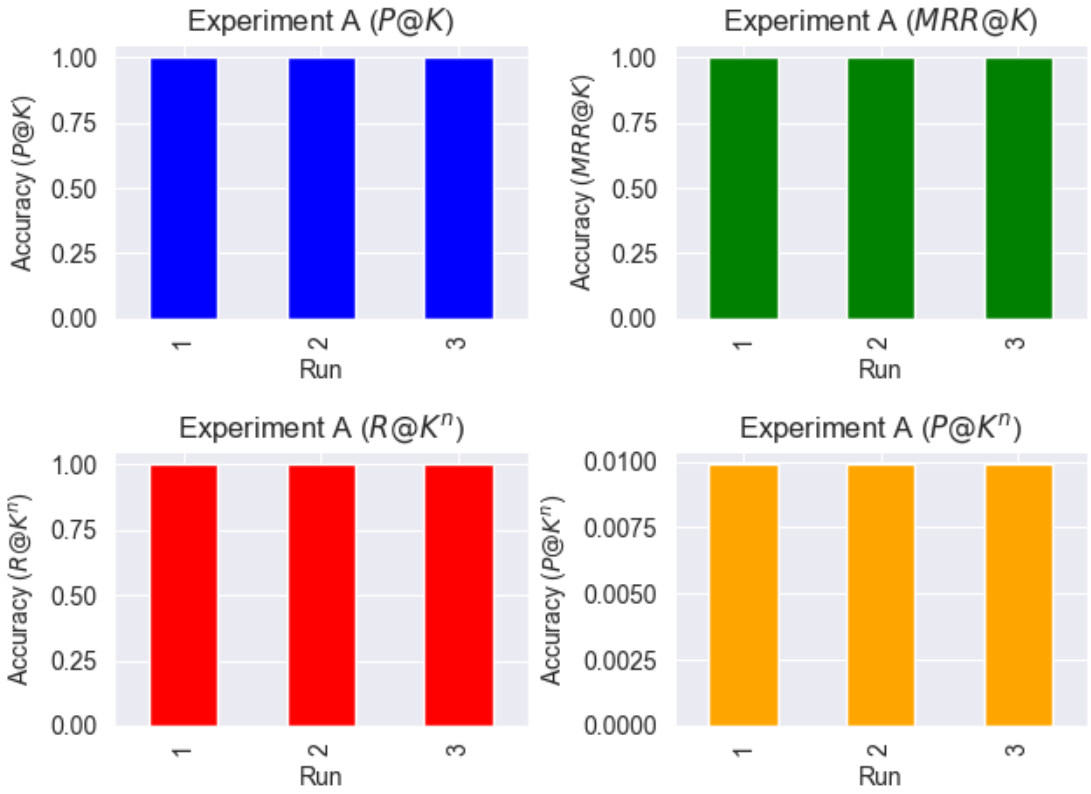


Figure 5.15.: Results of *Experiment A* for the data set *YooChoose* with $\mathcal{K} = 10$. Each bar represents a run of the experiment performed under different specifications of the hyperparameter RUN-MODE. The first, second, and third experimental runs are associated with STRUCTURAL, SESSION, and HISTORY, respectively. This graphical representation facilitates an analytical inspection of the effects induced by variations in the RUN-MODE hyperparameter.

5.5.2. Experiment B

In the data set *YooChoose*, it is impossible to compare the different negative sampling methods. This limitation is caused by the absence of item features necessary to compute the similarity between the item vectors.

However, in the absence of these features, the challenge is to develop alternative approaches that compensate for the unavailability of item features while still allowing for a meaningful comparison of the negative sampling methods. Methods based on other data features can be used for this purpose, or machine learning techniques to achieve

an implicit characterization of the elements. One possible approach within this model is to use the vectors from the embedding layer as a proxy for the features of the items. However, this approach is only feasible after initial training and could lead to considerable biases.

This thesis does not address negative sampling approaches based on similar item information or other techniques when item information is unavailable. This could be considered in future research. Additionally, this thesis does not explore the potential impact of this limitation on the performance of the model.

5.5.3. Experiment C

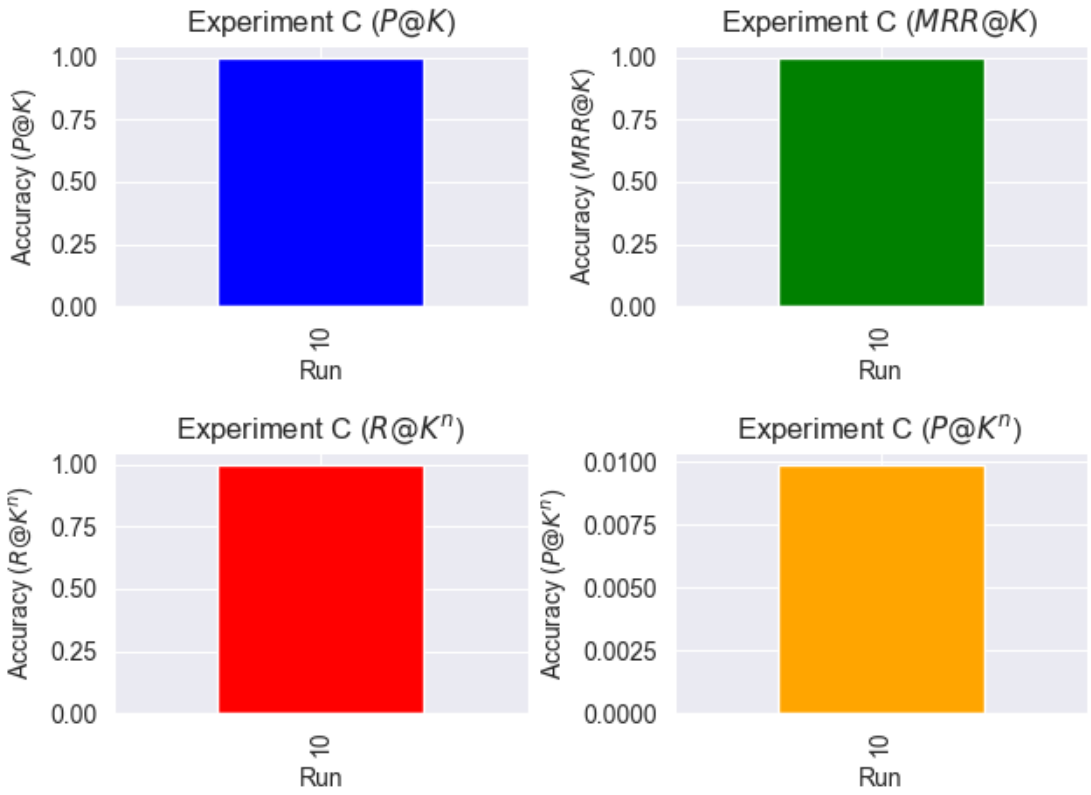


Figure 5.16.: Results of *Experiment C* for the data set *YooChoose* with $\mathcal{K} = 10$. In run 10, SESSION is applied while CATEGORY-SHIFT is not executed. The purpose is to explore the effect of this particular hyperparameter.

In this subsection, the results of *Experiment C* are presented, which are performed on

the *YooChoose* data set with $\mathcal{K} = 10$, as shown in Figure 5.16. The experimental setup is described in detail in subsection 5.1.3. The evaluation metrics used in this study are defined in subsection 2.5.2. The complete results are listed in Table A.17.

For *Experiment C*, the run mode is set to SESSION and the negative sample to RANDOM. The effect of category changes is only investigated for TGR-SES since it can be considered independently in this particular case. The results are as follows: $P@K = MRR@K = R@K^n = 0.9999$. These values show that this combination does not significantly degrade the precision of the recommendation. The results are consistent with those of the previous experiments.

In general, the results of this experiment indicate that the recommendation system achieves a high level of accuracy in generating recommendations.

5.5.4. Experiment D

Evaluation of various parameters for HISTORY has not been performed for this data set. This decision is due to the unique structure of the *YooChoose* data set, which consists only of session data. On the contrary, HISTORY requires sequential data, which indicates an order or sequence beyond the information contained in the session data. Therefore, using HISTORY is neither appropriate nor necessary in this context since no runs need to be associated with the data. This conclusion is further demonstrated in *Experiment A* of this data set. No discernible differences were found between HISTORY and SESSION, mainly because sessions cannot be combined.

5.5.5. Experiment E

In this section, the results of *Experiment E* are presented, which was performed on the *YooChoose* data set with $\mathcal{K} = 10$, as shown in Figure 5.17. The experimental setup is described in detail in subsection 5.1.5. The evaluation metrics used in this study are defined in subsection 2.5.2. The complete data set can be found in Table A.18.

The analysis of the empirical results shows that the best values for all metrics were achieved in both tests. It can be concluded that the recommendation system performed optimally under the test conditions. Thus, the system can generate accurate and relevant recommendations despite the limited number of items and users.

A significant aspect is that, regardless of hyperparameter oscillations, full accuracy in discriminating $R@K^n$ is achieved in both experimental tests. This is a robust proof of the resilience and effectiveness of the system in avoiding unwanted recommendations.

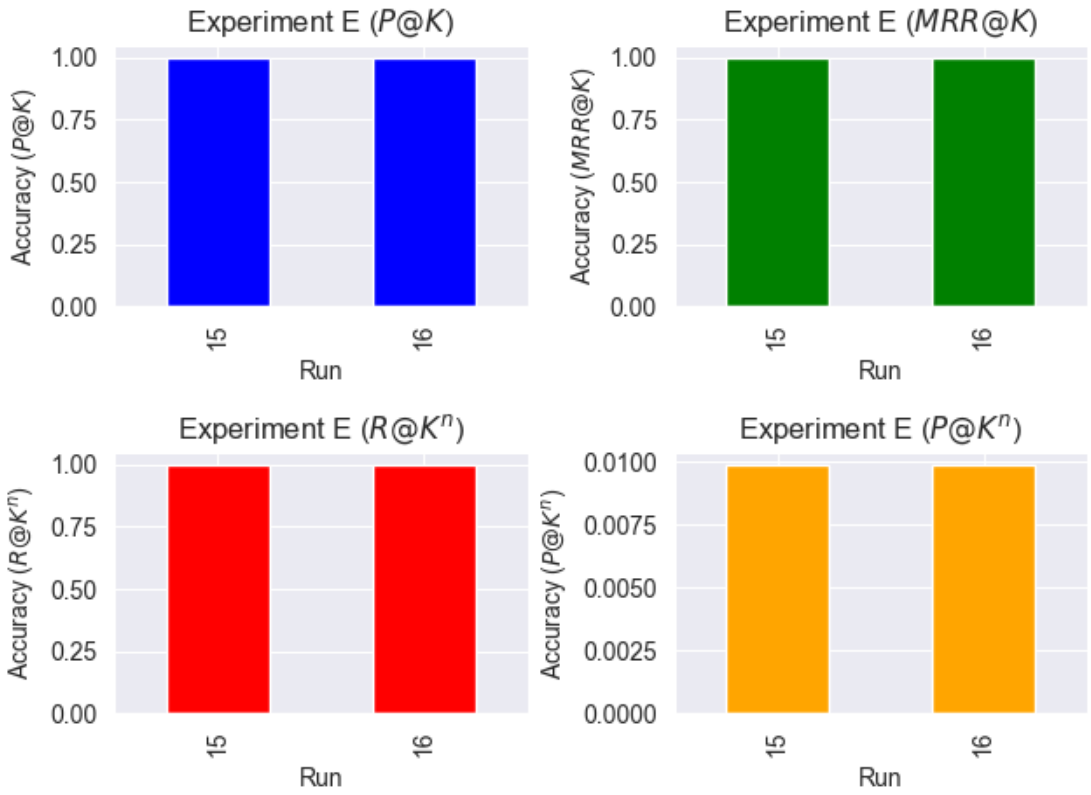


Figure 5.17.: Results of *Experiment E* for the data set *YooChoose* with $\mathcal{K} = 10$. In the graph, each bar represents a different experimental run performed with different configurations of the RUN-MODE hyperparameter. Specifically, runs 15 and 16 were run with the SESSION parameter, omitting USE-CATEGORY-SWITCH. Run 15 is designed to examine the sequence length parameter in separate analyses, denoted by MIN-COUNT-USER =MIN-COUNT-ITEM =3. Conversely, run 16 examined the sequence length with the configuration REMOVE-COMMON-USERS =REMOVE-COMMON-ITEMS =1. For scenarios not explicitly highlighted, the default hyperparameter settings were used. This figure allows the study of the effects associated with different sequence lengths.

Based on these empirical results, the optimal configuration for recommendation systems remains elusive, as it does not appear to have a discernible impact. This observation is particularly impressive because changes in the minimum number do not lead to significant changes, suggesting that even short sessions could be considered. These empirical findings provide a remarkable starting point for subsequent research. Although the dilemma of setting appropriate thresholds for these two parameters remains unresolved, this experiment provides valuable information, as the choice of MIN-COUNT-USER, MIN-COUNT-ITEM, REMOVE-COMMON-USERS, or REMOVE-COMMON-ITEMS significantly influences the results.

5.5.6. Conclusion

This section summarizes the experiments conducted on the *RetailRocket* data set. The first experiment examined the effect of hierarchical architecture on network prediction accuracy. Figure 5.13 shows the results of all experiments conducted on the data set *RetailRocket* with K = 10. The evaluation metrics used are defined in the background subsection 2.5.2.

Experiment A showed that using the SESSION mode in combination with MOST-SIMILAR produced the best results. The recommendation system achieved an impressive $P@K$ value of near 1, corresponding to a $MRR@K$ value of 1. On the contrary, the other two parameters of RUN-MODE, namely STRUCTURAL and HISTORY, showed lower performance values for the $R@K^n$ metric. Interestingly, the values for $P@K$ and $MRR@K$ were consistently close in all experiments, possibly indicating a network problem. However, a more detailed investigation of this issue is beyond the scope of this paper.

Experiment C examined the effect of CATEGORY-SHIFT on SESSION and found no positive effect. The accuracy of the recommendation system remained unchanged.

Experiment E used optimal hyperparameter configurations and highlighted the significant effects of choosing the MIN-COUNT-USER, MIN-COUNT-ITEM, and REMOVE-COMMON-USERS,REMOVE-COMMON-ITEMS of interactions per node on the performance of the recommendation system. No performance changes were observed based on the metrics used.

The experiments in the *YooChoose* data set showed that SESSION provided the most accurate recommendations. They achieved high values for the metrics $P@K$, $MRR@K$, and $R@K^n$, demonstrating the accurate identification of relevant recommendations.

A correlation matrix is synthesized to quantify the interrelationships among the

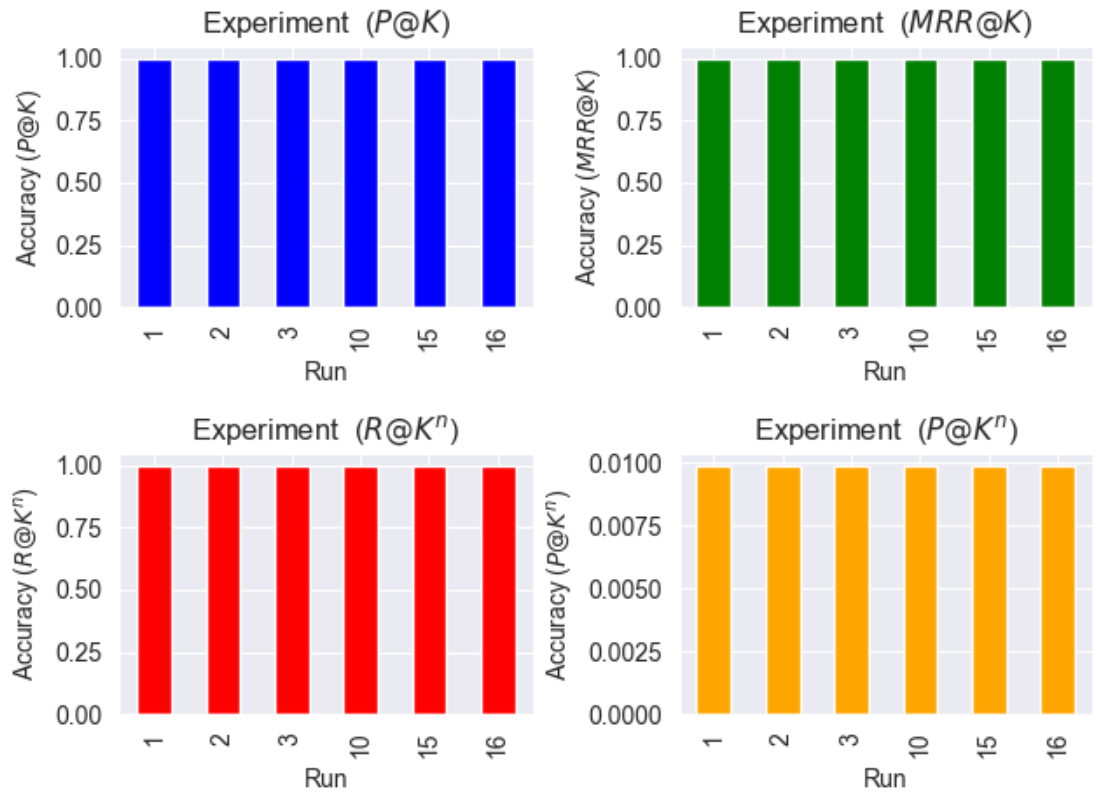


Figure 5.18.: Results of the experiments for the data set *YooChoose* with $\mathcal{K} = 10$

hyperparameters, as shown in Figure 5.19. The matrix's correlation coefficients indicate the associations' strength and direction. A coefficient of 1 represents a perfect positive correlation, while -1 represents a perfect negative correlation. Numbers close to 0 indicate minimal or negligible correlation. In this dissertation, due to the lack of sequences, TGR-HIS was not used, so there is a corresponding lack of data on USE-SESSION-TIME-SAVE within the correlation matrix.

A robust positive correlation between the metrics and MIN-COUNT-USER, MIN-COUNT-ITEM,REMOVE-COMMON-USERS, and REMOVE-COMMON-ITEMS. However, since these parameters were only examined in a duo of experiments, the comprehensiveness of these correlations cannot be determined conclusively. Given this inconsistency with the derived results, the author postulates that these parameters negatively influence their investigation.

Regarding the hyperparameter USE-CATEGORY-SWITCH, there is a strong positive correlation. This means that the experimental results indicate a significant influence of

USE-CATEGORY-SWITCH, suggesting its nonuse.

It is crucial to emphasize that this analysis is based on the available data, which requires further research and validation to understand the relationships between the metrics better. It is also important to emphasize that correlations alone do not prove causality. Additional experimental research and analysis are mandatory to confirm the essential relationships and embeddings of the measured variables.

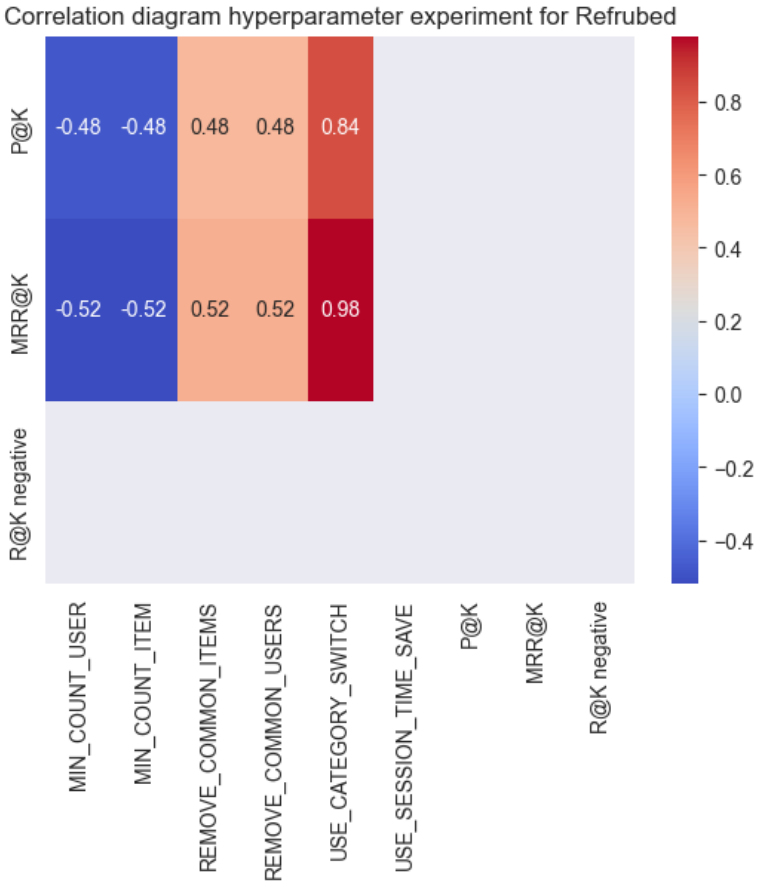


Figure 5.19.: Correlation diagram of the hyperparameters for the data set *YooChoose* with $\mathcal{K} = 10$

5.5.7. Comparison with other works

The Table 5.9 presents the results of different models to evaluate the recommendation systems using metrics $P@20$ and $MRR@20$. Models are compared with different runs of the TGR model. The results are generally taken from [Hou+22] to ensure consistency

Model	Reference	P@20	MRR@20
FPMC	section 5.2	-	-
GRU4Rec	section 5.2	60.78	27.27
NARM	section 5.2	61.67	27.82
SR-GNN	section 5.2	61.84	28.15
NISER+	section 5.2	62.99	28.98
LESSR	section 5.2	62.89	28.59
SGNN-HN	section 5.2	62.49	28.24
SASRec	section 5.2	63.55	28.63
GC-SAN	section 5.2	63.24	29.0
CL4Rec	section 5.2	63.61	28.73
CORE	section 5.2	64.61	28.24
TGR (Experiment: 10)	0.99	0.99	
TGR (Experiment: 15)	1	1	
TGR (Experiment: 16)	1	1	
TGR (Experiment: 20)	0.99	0.99	

Table 5.8.: Results of the experiments for the data set *YooChoose*, Results based on: [Hou+22]

in interpreting the results, referencing individual models in the appropriate sections. Descriptions of the experiments can be found in the appropriate subsections of this section or, more generally, in section 5.1.

Run 20 is similar to run 10, except that no minimum or maximum filtering of nodes is applied to ensure a fair comparison with the other models, and it is based on *YooChoose 1/64*, since otherwise a computing time would have been needed, which cannot be done within the scope of this thesis.

The models compared include *FPMC*, *GRU4Rec*, *NARM*, *SR-GNN*, *NISER+*, *LESSR*, *SGNN-HN*, *SASRec*, *GC-SAN*, *CL4Rec*, and *CORE*. These models have received considerable attention in previous studies, and their performance for various scenarios and data sets has been extensively investigated.

The Table 5.9 demonstrates that the TGR consistently achieves impeccable values for $P@20$ and $MRR@20$, regardless of whether the minimum number of interactions per node is reduced, outperforming other GNN-based models. For example, in run 10, the $P@20$ value is 0.99, in contrast to the value of $P@20 = 52.89$ obtained for *CORE*. This trend remains consistent even in run 20, where an $P@20 = 0.99$ is recorded.

These data suggest that the TGR model matches or exceeds the recommendation accuracy of established GNN-based models when no minimum number of nodes is used. However, a direct comparison between these models is often not feasible, as TGR was explicitly designed for internal comparison with the *Refurbed* data set and is therefore not optimized for small session sizes, a feature that has been touted as beneficial in other studies [Wu+20a]. Despite these limitations, a score of $P@20 = 0.99$ was still obtained, a surprising and encouraging result.

Future research should confirm the reproducibility of these findings and examine the possible impact of δ between $P@K$ and $MMR@K$. The author suspects this discrepancy may introduce bias into the results, and further investigation is warranted.

In summary, the analysis shows that the TGR model has effectively addressed the problem of the *YooChoose* data set, outperforming traditional GNN-based models regarding precision and accuracy. However, the relatively small δ between $P@K$ and $MMR@K$ requires further investigation before concluding. It is important to note that this comparison focuses only on positive metrics. An analysis that includes the metrics $R@K^n$ would be beneficial, as it could reveal potential advantages of the TGR model or expose the strengths and weaknesses of the competing models.

5.5.8. Open topics

This section presents a list of research topics that should be addressed in future research based on the *YooChoose* data set.

$MRR@K - P@K$ Observations suggest that when an item is hit in $P@K$, it is unlikely to have a position of 1. A similar trend is expected for the corresponding values of $MRR@K$ [Wu+20a; Gao+21]. These findings may indicate possible inconsistencies in the model that require further investigation.

Criterion The evaluation of alternative criteria that may better fit this model has not yet been performed. Future research in this direction would be beneficial.

Negative sampling There is a need to explore alternative approaches to the meaningful comparability of negative sampling methods without item characteristics. Consideration could be given to embedding layer vectors, which may have a bias. [Min/Max number of edges] More research is needed to determine how many edges each node (user or item) requires in the bipartite graph so that this model retains its ability to generate recommendations. Similarly, more research is needed to determine if a maximum number of edges reduces the predictive quality, as the prediction of these is more likely than more unknown nodes.

use-session-time-save or use-category-switch It is unclear whether both parameters offer similar improvements and which is most appropriate for this data set. More detailed research is needed in this regard. Also, the functions behind these methods could be investigated to see if similar approaches are more appropriate.

embedding-dimension-for-item for use-dimension-for-item The optimal hidden size for this data set has not been adequately studied. More research is needed to determine the most appropriate hidden size for this particular task. This is also true for the hyperparameters that control user embeddings (EMBEDDING-DIMENSION-FOR-USER for USE-DIMENSION-FOR-USER).

In general, uncertainties remain regarding the suitability of this model and possible avenues for improvement. Thorough research is needed to determine the threshold beyond which the model's ability to detect and predict sessions diminishes. Overall, it can be declared that TGR is suitable for the *YooChoose* data set and gives perfect results in certain situations with a minimum number of items in a session.

5.6. Diginetica

This section describes in detail the experiments and their results obtained with the *Diginetica* dataset, which is described in section 4.2.

5.6.1. Experiment A

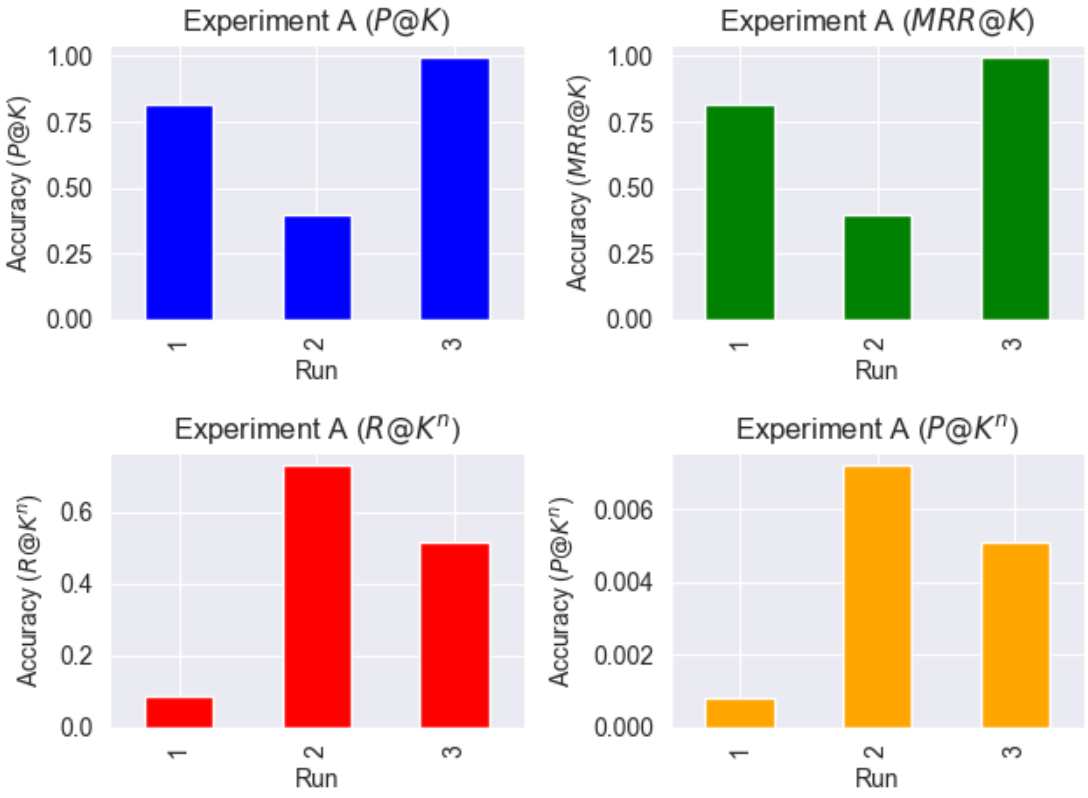


Figure 5.20.: Results of *Experiment A* for the data set *Diginetica* with $\mathcal{K} = 10$. Each bar represents a run of the experiment performed under different specifications of the hyperparameter RUN-MODE. The first, second, and third experimental runs are associated with STRUCTURAL, SESSION, and HISTORY, respectively. This graphical representation facilitates an analytical inspection of the effects induced by variations in the RUN-MODE hyperparameter.

In Figure 5.20, the results of *Experiment A* for the data set *Diginetica* with $\mathcal{K} = 10$ are shown. The experimental setup is described in subsection 5.1.1. The metrics used are defined in subsection 2.5.2. Result data are available at Table A.6.

The results indicate that RUN-MODE, combined with the parameter SESSION and negative sampling MOST-SIMILAR, achieves the best performance. The recommendation system achieves an excellent $P@K$ value of 0.9429 and an almost identical $MRR@K$ value of 0.9420. These elevated values suggest that the system can generate a significant proportion of correct recommendations within the top $\mathcal{K}$ recommendations and that the ranking of correct recommendations is nearly optimal.

In this experiment, it is evident that STRUCTURAL exhibits relatively favorable accuracy in predicting recommendations, albeit with limitations in identifying relevant negative examples. On the other hand, using RUN-MODE in conjunction with SESSION shows improved detection of relevant negative examples, albeit at the expense of overall prediction accuracy. On the contrary, HISTORY achieves nearly flawless prediction accuracy and excels at detecting relevant negative examples, although at a slightly lower recommendation ranking compared to the other RUN-MODE. The optimal results vary depending on the metric used.

In particular, the results indicate that the combination of STRUCTURAL with MOST-SIMILAR produces remarkable results. The values obtained for $P@K$ and $MRR@K$ are 0.8165, which indicates a comparatively accurate prediction capability of the recommendation system and a robust ranking of recommended items. However, the problematic relationship between $P@K$ and $MRR@K$ still exists and needs further investigation in future research. In contrast, the values for $R@K^n$ and $P@K^n$ are notably low at 0.0878 and 0.0008, respectively. This suggests that the system has difficulty distinguishing positive feedback from randomly selected negative examples and struggles to minimize the misclassification of irrelevant items.

In comparison, SESSION provides lower scores for $P@K$ and $MRR@K$ at 0.4, indicating a lower prediction precision and a weaker recommendation ranking for the recommendation system in this case. However, the scores for $R@K^n$ and $P@K^n$ are significantly higher than in other runs of this experiment, at 0.7289 and 0.0072, respectively. This suggests that the system can identify many relevant negative examples, albeit at the expense of overall prediction accuracy.

In contrast, HISTORY shows almost excellent accuracy with $P@K = MRR@K = 0.9952$. This indicates that the recommendation system in this particular mode can make highly accurate predictions and effectively sort the recommended items. The $R@K^n = 0.5176$ score outperforms the results of STRUCTURAL, indicating that the system can detect a reasonable number of relevant negative examples, although not as effectively as in SESSION.

These results generally confirm the previously expected results and the relevance of the hyperparameters chosen to determine the performance of GNN-based recommendation systems. However, further investigations are necessary. In particular, it is interesting to investigate why the TGR-SES results show unexpectedly weak performance, especially considering that the TGR-HIS results are based on them. One possibility could be that the items only have a feature depth of 12. At the same time, the hidden size of the session graph ($\mathcal{G}^s$, described in subsection 3.4.2 is much larger by default at 200. This aspect deserves further investigation.

5.6.2. Experiment B

In Figure 5.21, the results of *Experiment B* for the data set *Diginetica* with $\mathcal{K} = 10$ are shown. The experimental setup is described in subsection 5.1.2. The metrics used are defined in subsection 2.5.2. Result data are available in Table A.7. The results from runs 1-3 are provided in subsection 5.6.1 and compared with the other negative sampling methods in this section.

Runs 1, 4, and 7 apply to the STRUCTURAL parameter. Interestingly, the RANDOM negative sampling method provides the best results in this setup. While the values for positive feedback are very similar, suggesting a negligible impact, the more intriguing aspect is the impact on the metrics, where the negative feedback scores are $P@K = 0.8384$ and $R@K^n = 0.1821$. While the values for positive feedback are very similar, suggesting a negligible impact, the more intriguing aspect is the impact on metrics that account for negative feedback. There is more than a twofold improvement in the results, although they generally remain inferior to the positive metrics.

Runs 2, 5, and 8 investigate the SESSION parameter. Surprisingly, the MOST-SIMILAR and LESS-SIMILAR sampling methods do not meet the author's expectations, performing worse than the other two RunModes. It is interesting to note that RANDOM performs worse than STRUCTURAL, with $P@K = 0.2707$, comparable to the performance of HISTORY. This suggests that STRUCTURAL randomly selects more appropriate negative examples than the other two run modes. More research is needed, especially when considering metrics that account for negative examples. In this experiment, 100 randomly selected negative targets were considered. If the author assumes that randomly selected items give worse results in SESSION, they would also give worse results as negative examples. However, this contradicts the excellent results of $R@K^n = 0.7289$ compared to $R@K^n = 0.1821$ in STRUCTURAL, where the best result is obtained in this comparison. This discrepancy

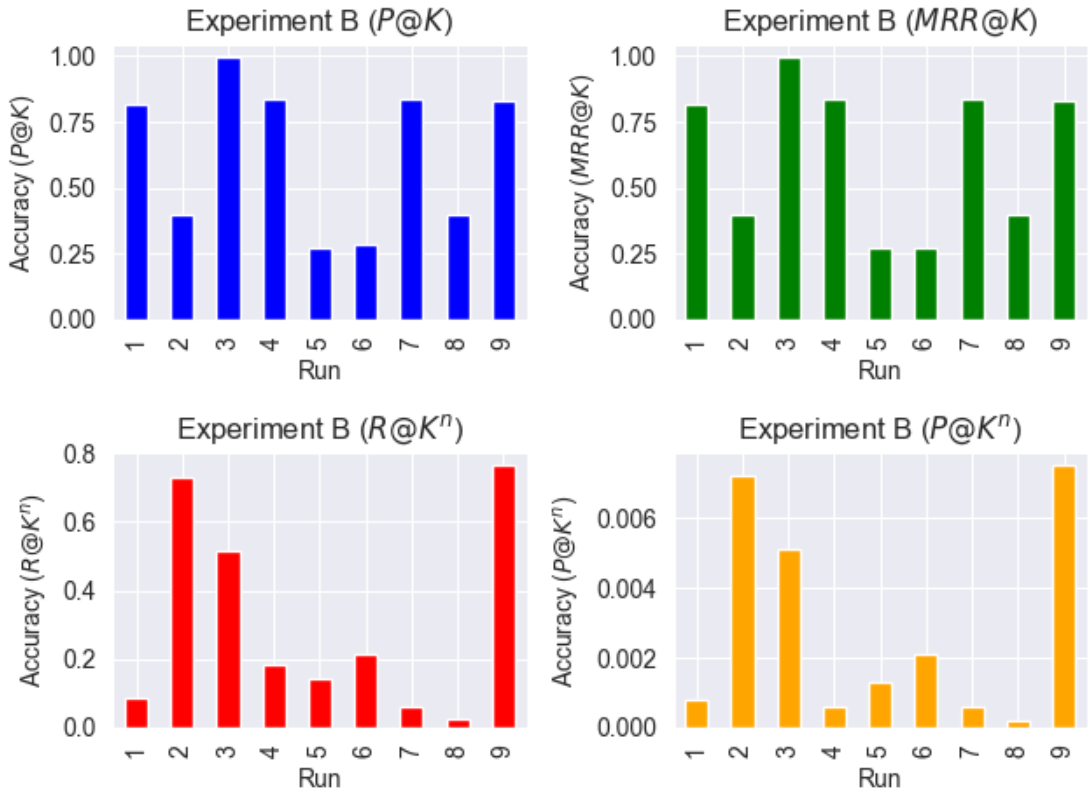


Figure 5.21.: Results of *Experiment B* for the data set *Diginetica* with $\mathcal{K} = 10$. Each bar in the graph represents a unique run of the experiment. These runs vary according to the different specifications of the hyperparameter RUN-MODE, which is used in conjunction with NEGATIVE-SAMPLER. The first, second, and third runs are associated with STRUCTURAL, SESSION, and HISTORY, respectively. In these runs, they are combined with MOST-SIMILAR. The fourth, fifth, and sixth runs again use STRUCTURAL, SESSION, and HISTORY. However, in these runs, they are paired with RANDOM. Lastly, the seventh, eighth, and ninth runs utilize STRUCTURAL, SESSION, and HISTORY, respectively. In these instances, they are paired with LESS-SIMILAR. This figure simplifies the analysis of the effects induced by variations in the RUN-MODE hyperparameter when combined with different NEGATIVE-SAMPLER options.

needs to be investigated further. Furthermore, the unusually high results of run two could be explained by using randomly selected items that positively match the metric. It is important to note that this study did not include repeated model training. Therefore future work is needed to replicate these results to explain the differences.

The HISTORY runs are described in 3, 6, and 9. Here, it can be seen that MOST-SIMILAR gives the best results with $P@K = 0.9952$, while LESS-SIMILAR also provides good scores with $P@K = 0.8384$. Interestingly, the HISTORY parameter with RANDOM ($P@K = 0.2707$ and $R@K^n = 0.1405$) fails to correctly predict a target compared to the other two runs, which was unexpected by the author. However, $R@K^n = 0.7650$ still gives favorable results.

In summary, when considering only positive targets, the combination of HISTORY and MOST-SIMILAR gives the best results. However, when negative targets are also considered, LESS-SIMILAR performs significantly better. It remains to be seen whether these changes, combined with other hyperparameters, yield similar results in subsequent experiments.

5.6.3. Experiment C

In Figure 5.22, the results of *Experiment C* for the *Diginetica* data set with $\mathcal{K} = 10$ are presented. The experimental setup is described in subsection 5.1.3. The metrics used are defined in subsection 2.5.2. Result data can be found in Table A.8.

In this experiment, the run mode was set to SESSION, and the negative sample was set to MOST-SIMILAR. The influence of CATEGORY-SHIFT was only examined for TGR-SES since it can only be considered separately. The results are as follows: $P@K = 0.4344$, $MRR@K = 0.4344$, $R@K^n = 0.4213$, and $P@K^n = 0.0042$. These scores indicate that this combination does not significantly improve the precision of the recommendation. Furthermore, it can be observed that the negative metrics are particularly affected and show a significant negative impact. The results of this experiment are less robust than those obtained with CATEGORY-SHIFT, which is consistent with the expectation that the degradation of the category change functionality may have a negative impact, as discussed earlier in the data analysis for this data set.

In general, the findings of this experiment indicate that the recommendation system has low accuracy in recommendation generation and a tendency toward poor recommendation ordering. Future experiments will determine if this is also true for the SESSION in general.

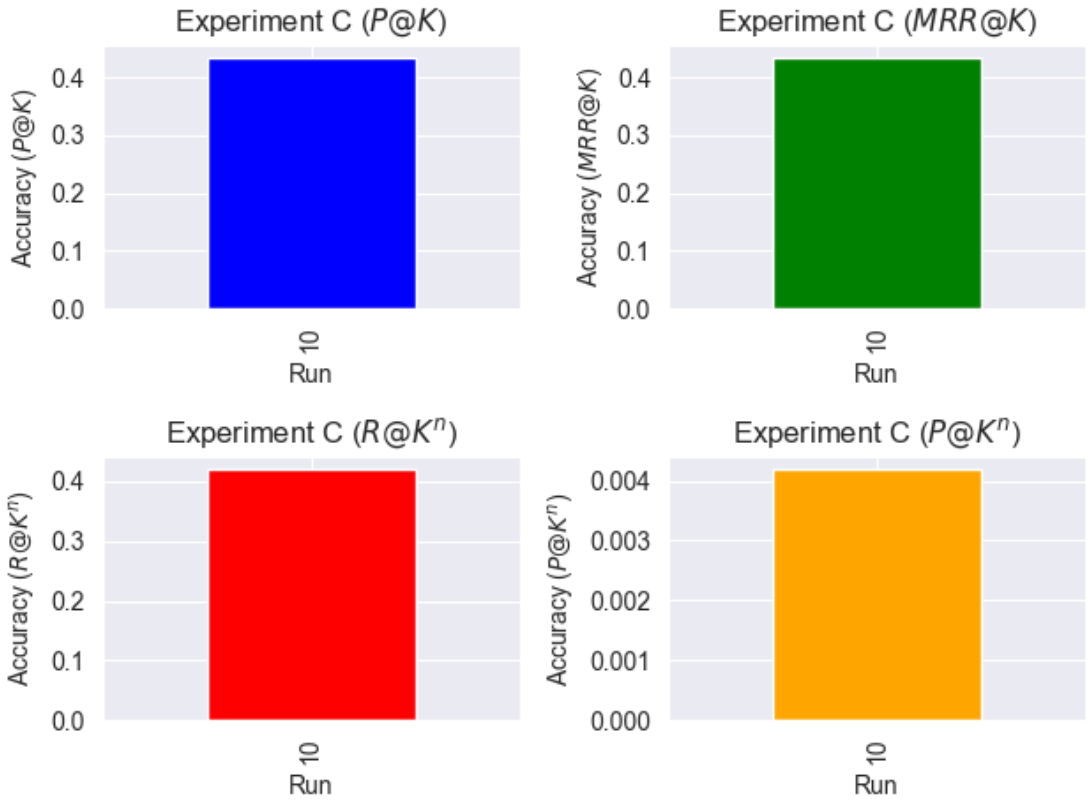


Figure 5.22.: Results of *Experiment C* for the data set *Diginetica* with $\mathcal{K} = 10$. In run 10, SESSION is applied while CATEGORY-SHIFT is not executed. The purpose is to explore the effect of this particular hyperparameter.

5.6.4. Experiment D

In Figure 5.23, the results of *Experiment D* for the data set *Diginetica* with $\mathcal{K} = 10$ are shown. The experimental setup is described in subsection 5.1.4. The metrics used are defined in subsection 2.5.2. Result data are available in Table A.9.

The first combination is MOST-SIMILAR without using CATEGORY-SHIFT and USE-SESSION-TIME-SAVE. Here, the values of $P@K = MRR@K = 0.8078$ were obtained. The results indicate that the model generally has no difficulty generating relevant recommendations across the entire set of experiments. However, compared to other runs in *Experiment D*, other runs produced higher accuracy. In particular, looking at $R@K^n = 0.0971$, it is clear that this model cannot prioritize the positive target over the negative examples, which indicates that the model could not capture the

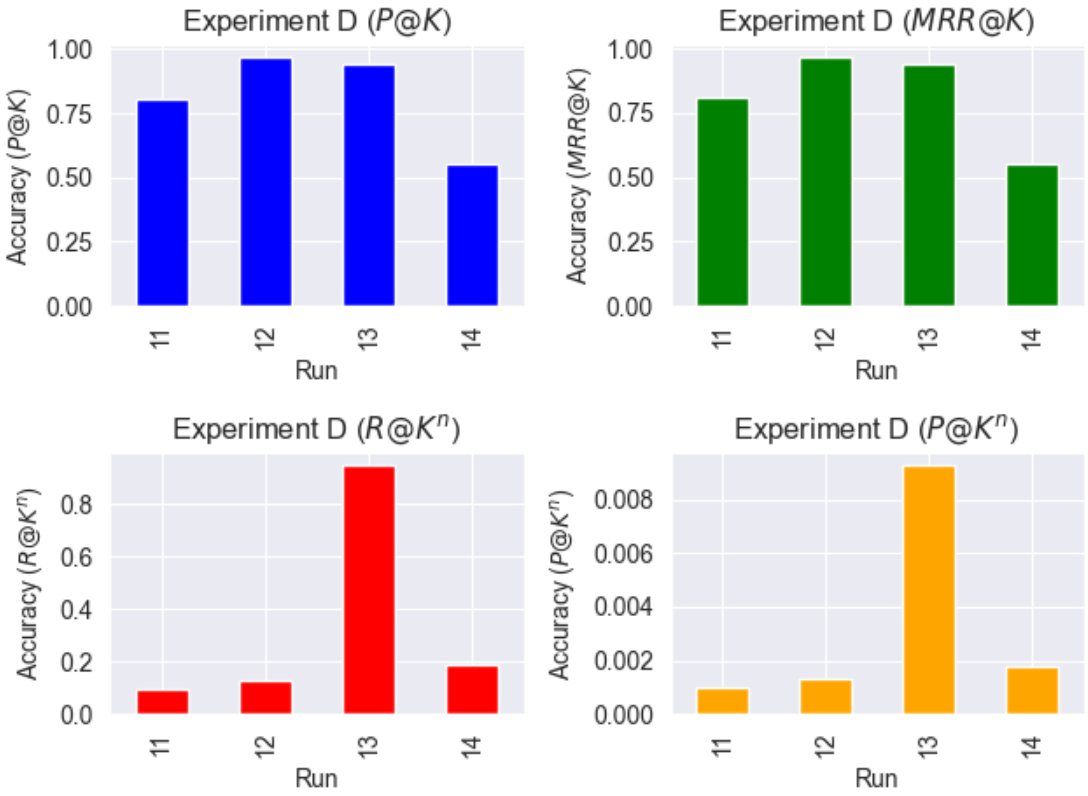


Figure 5.23.: Results of *Experiment D* for the data set *Diginetica* with $\mathcal{K} = 10$. Runs 11 through 14 apply HISTORY while excluding both CATEGORY-SHIFT and USE-SESSION-TIME-SAVE. The purpose is to investigate the effect of the USE-SESSION-TIME-SAVE hyperparameter. Specifically, runs 11, 12, and 14 do not use CATEGORY-SHIFT and USE-SESSION-TIME-SAVE. However, run 12 uniquely includes min-max normalization. In contrast, run 13 uses CATEGORY-SHIFT but excludes USE-SESSION-TIME-SAVE. Finally, run 14 uses LESS-SIMILAR, while all other runs of this experiment use MOST-SIMILAR.

complexity of the data and adequately model the differences between positive and negative examples. However, compared to the CATEGORY-SHIFT enabled, it shows that $P@K = MRR@K = 0.9415$, indicating that this configuration can generate better recommendations. The high value of $R@K^n = 0.9489$ indicates that most negative examples were correctly identified, which could indicate that using CATEGORY-SHIFT in conjunction with GNN allows better modeling of the interactions and thus improves the quality of the recommendations. In comparison, LESS-SIMILAR yields worse values in all

metrics: $P@K = 0.5542, MRR@K = 0.5536, R@K^n = 0.1863$. These moderate values indicate that only some negative examples were correctly identified. However, compared to the first run of this experiment, it appears that the number of examples incorrectly identified as negative has been somewhat reduced. Overall, however, this configuration shows limited modeling capability and reduced ability to capture interactions in the data adequately.

These results highlight the significant impact of hyperparameter selection on the performance of the recommendation system. In particular, varying the usage of USE-SESSION-TIME-SAVE resulted in notable variations in metric scores, especially compared to previous experiments on this data set. However, it remains unclear whether the problem is solely due to the chosen method or whether manipulating session embeddings with timestamps to establish temporal distinctiveness for the attribution layer is inappropriate. Further investigation of this issue is beyond the scope of this paper. On the other hand, the use of USE-CATEGORY-SWITCH has shown positive effects, significantly improving metrics that account for negative feedback.

5.6.5. Experiment E

In Figure 5.24, the results of *Experiment E* for the data set *Diginetica* with $\mathcal{K} = 10$ are shown. The experimental setup is described in subsection 5.1.5. The metrics used are defined in subsection 2.5.2. The results are available in Table A.10.

This experiment used the most effective hyperparameters for TGR-SES and TGR-HIS. Runs 15 and 16 are used with TGR-SES, MOST-SIMILAR, and CATEGORY-SHIFT disabled. Runs 17 and 18 used TGR-HIS with MOST-SIMILAR disabled and without USE-SESSION-TIME-SAVE, but with USE-CATEGORY-SWITCH enabled. These runs investigated whether TGR-SES performs better with a different number of data points and whether the previous optimal configuration for this data set remains equally effective with a different number of edges per node.

Runs 15 and 17 focused on evaluating five minimal items and users. The results showed that both TGR-SES and TGR-HIS achieved $P@K = MRR@K = 1$, suggesting that the model accurately identified the most relevant items and positioned them among the top recommendations K. However, it is important to note that due to the reduced number of nodes, only 200 edges could be examined, resulting in a lower probability that items will be included in the top recommendations $\mathcal{K}$. Examining the negative values, obtain $P@K^n = 0.7700$ for TGR-SES and $P@K^n = 0.8850$ for TGR-HIS. This implies that

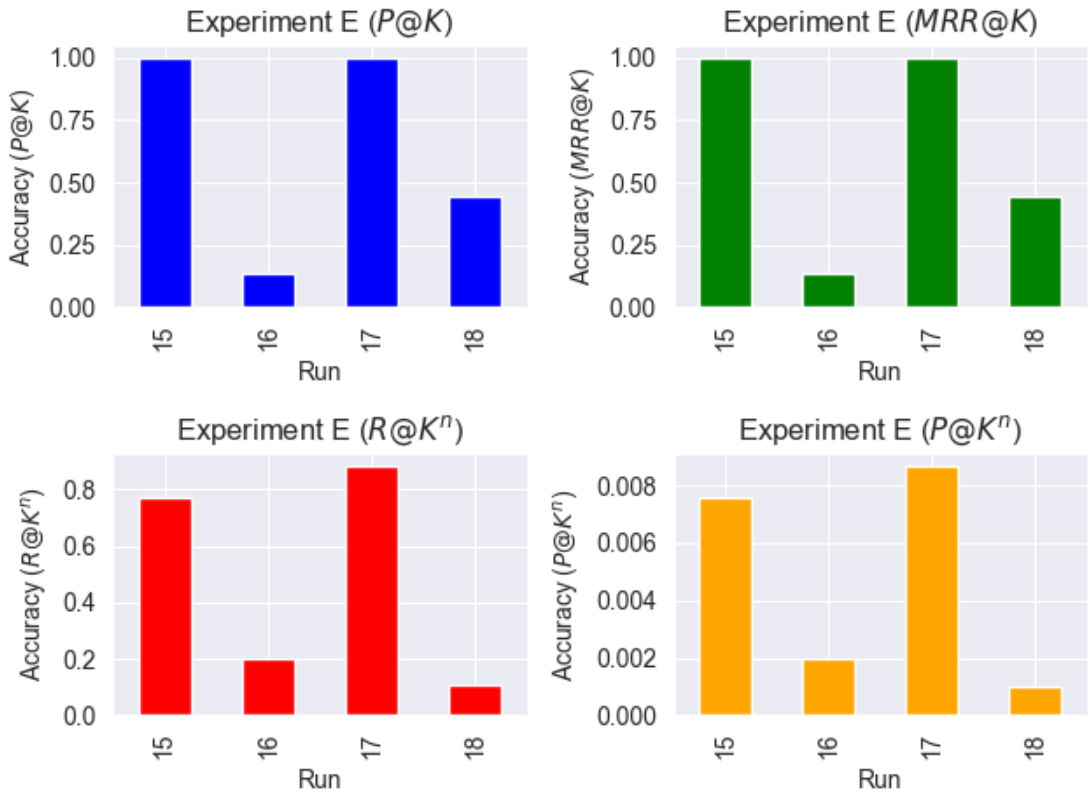


Figure 5.24.: Results of *Experiment E* for the data set *Diginetica* with $\mathcal{K} = 10$. The graph illustrates various experimental runs. Runs 15 and 16 utilized the SESSION parameter, excluding USE-CATEGORY-SWITCH, while runs 17 and 18 employed the HISTORY parameter, excluding both USE-CATEGORY-SWITCH and USE-SESSION-TIME-SAVE. Additionally, runs 15 and 17 were designed to study the sequence length parameter, represented by MIN-COUNT-USER =MIN-COUNT-ITEM =3. In contrast, runs 16 and 18 explored sequence length with the configuration REMOVE-COMMON-USERS =REMOVE-COMMON-ITEMS =1. Default hyperparameter settings were applied in scenarios not explicitly mentioned.

longer sequences, which may provide more information about user preferences, may have a positive effect. It was also observed that TGR-HIS remained the superior execution mode.

In contrast, Runs 15 and 17 removed 0.1 of the items and users most frequently used. It was found that TGR-SES achieved $P@K = MRR@K = 0.13733, P@K^n = 0.2217$, while TGR-HIS achieved $P@K = MRR@K = 0.4426, P@K^n = 0.1075$. These results suggest that the precision of selecting relevant elements in the recommendations was lower in this case.

Based on these results, the optimal configuration for TGR in this scenario includes a minimum of three items, three items, a maximum of 0.05 items, and 0.05 users. This finding is particularly noteworthy because the minimum number of 5 does not show significant changes, suggesting that short sessions can be considered. These results serve as a starting point for further research. The question of determining appropriate thresholds for these two parameters remains open. However, it can be addressed through this experiment as the choice of minimum and maximum numbers significantly affects the results.

5.6.6. Conclusion

This section summarizes the experiments conducted on the *Diginetica* data set. The Figure 5.25 shows the results of all experiments performed on the data set *Diginetica* with $\mathcal{K} = 10$. The evaluation metrics used are defined in subsection 2.5.2.

Experiment A showed that the use of the HISTORY mode in combination with MOST-SIMILAR yielded the best results. The recommendation system achieved an impressive $P@K$ of 0.9629, closely matching the $MRR@K$ of 0.9629. On the contrary, the other two parameters for RUN-MODE, namely STRUCTURAL and HISTORY, showed lower performance values. It is interesting to note the proximity of the $P@K$ and $MRR@K$ values across all experiments, which may indicate a potential network problem. However, a more detailed investigation of this issue is beyond the scope of this thesis. *Experiment B* compared negative sampling methods for the three different RUN-MODE parameters. The results showed that combining MOST-SIMILAR with HISTORY produced the best results while combining RANDOM with SESSION produced the worst results. The choice of negative sampling had a significant impact on evaluation metrics.

Experiment C examined the effect of CATEGORY-SHIFT on SESSION and showed that it had no positive influence. The recommendation system showed low accuracy and an

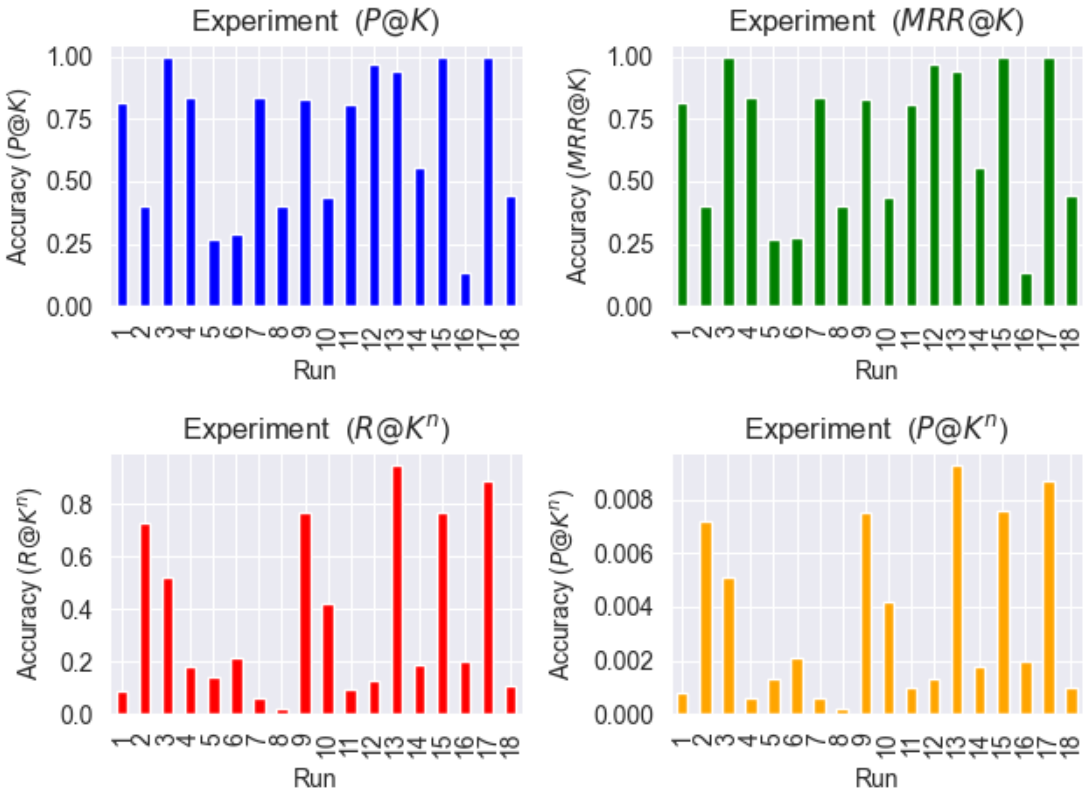


Figure 5.25.: Results of the experiments for the data set *Diginetica* with $\mathcal{K} = 10$

inferior recommendation order.

Experiment D investigated the effect of different hyperparameters on the performance of the recommendation system. It showed that negative sampling positively affected the evaluation metrics, and the choice of the hyperparameter USE-SESSION-TIME-SAVE resulted in significant variations in the metric values.

Experiment E used the best configurations of the hyperparameters and highlighted the significant impact of the choice of the minimum and maximum number of interactions per node on the performance of the recommendation system.

In summary, the mode HISTORY combined with the negative sampling method MOST-SIMILAR had the most significant impact on the performance of the recommendation system. On the contrary, the change in the category had no positive effect. The choice of hyperparameters, especially negative sampling in run mode, also significantly impacted the metric values.

In Figure 5.26, a correlation matrix was constructed to analyze the relationships

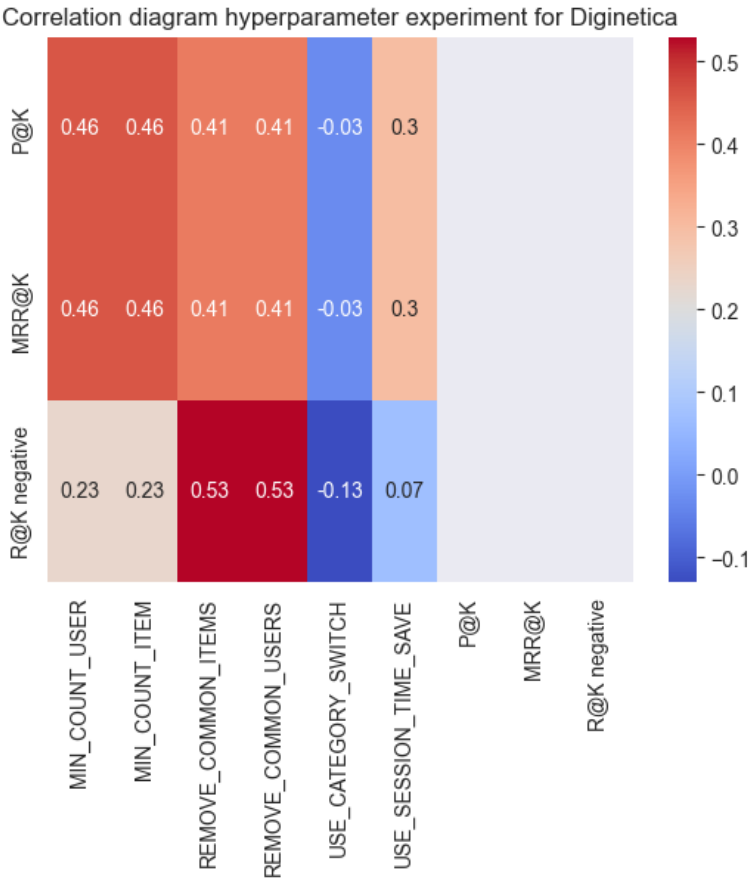


Figure 5.26.: Correlation diagram of the hyperparameters for the data set *Diginetica* with $\mathcal{K} = 10$

between various hyperparameters. The matrix's correlation coefficients indicate the strength and direction of the correlations, with a coefficient of 1 indicating a perfect positive correlation and -1 indicating a perfect negative correlation. Values close to 0 indicate negligible or minimal correlation.

MIN-COUNT-USER or MIN-COUNT-ITEM show a relatively strong positive correlation, which is less pronounced for REMOVE-COMMON-USERS or REMOVE-COMMON-ITEMS. However, it is important to note that since these parameters were only examined in two experiments, it is impossible to determine the extent of these correlations conclusively.

Regarding the hyperparameter USE-CATEGORY-SWITCH, a moderate negative correlation is observed. This suggests that the data do not support the assumption of a strong influence made in the experiment. Similarly, a more positive correlation is observed for

USE-SESSION-TIME-SAVE.

It is important to underline that this analysis is based on the available data, and further investigation and verification are required to understand the relationships between the metrics better. It is also important to note that correlations alone do not establish causality. Additional experimental research and analysis are required to confirm the measured variables' underlying relationships. However, the number of minimum elements per node appears to have the greatest influence, as supported by other studies [Wu+20a]. The feasibility and effectiveness of using different strategies is a matter of debate and warrants further exploration in future research.

5.6.7. Comparison with other works

Model	Reference	P@20	MRR@20
FPMC	section 5.2	31.83	8.79
GRU4Rec	section 5.2	45.43	14.77
NARM	section 5.2	47.68	15.58
SR-GNN	section 5.2	48.76	16.93
NISER+	section 5.2	51.23	18.32
LESSR	section 5.2	48.8	16.96
SGNN-HN	section 5.2	50.89	17.25
SASRec	section 5.2	49.86	17.19
GC-SAN	section 5.2	50.95	17.84
CL4Rec	section 5.2	50.03	17.26
CORE	section 5.2	52.89	18.58
TGR (Run: 12)		0.97	0.96
TGR (Run: 13)		0.94	0.94
TGR (Run: 17)		0.44	0.44
TGR (Run: 20)		0.38	0.37

Table 5.9.: Results of the experiments for the data set *Diginetica*, Results based on: [Hou+22]

The Table 5.9 presents the results of different models to evaluate the recommendation systems using metrics P@20 and MRR@20. Models are compared with different runs of the TGR model, which achieved different performance values in experiments. The results

were obtained from Hou et al. [Hou+22] to maintain consistency while interpreting the outcomes, and each model is referenced in its respective section. In run 20, there was no application of minimum or maximum filtering of nodes to ensure a fair comparison with the other models, unlike run 13.

The models compared include *FPMC*, *GRU4Rec*, *NARM*, *SR-GNN*, *NISER+*, *LESSR*, *SGNN-HN*, *SASRec*, *GC-SAN*, *CL4Rec*, and *CORE*. These models have received considerable attention in previous studies, and their performance for various scenarios and data sets has been extensively investigated. It is important to recognize that most of the models used for comparison address the *cold-start* problem only indirectly, using advanced techniques such as GNNs, self-learning mechanisms, RNNs, and latent factor modeling [RFS10; Hid+16; Li+17; Wu+19; Gup+21; CW20]. These techniques make it possible to model relationships between users and items, identify relevant patterns in the data, and make predictions based on general features rather than specific interactions, which could address *cold-start* challenges [Wu+20a; Gao+21; Wan+21b]. For example, *CORE* solves this problem by introducing the idea of meta-optimization into recommendation scenarios [Hou+22]. The core idea is to learn global, shared meta-initialization parameters for all users and quickly adapt them to local parameters for each user. This enables fast adaptation to new users with learned prior knowledge [Hou+22].

As shown in the table, the TGR generally exhibits lower values for $P@20$ and $MRR@20$ when the number of minimum interactions per node is reduced compared to the other models based on GNN. In run 17, the value of $P@20$ is 0.97, while for *CORE*, it is $P@20 = 52.89$. However, the perspective on the results changes when looking at run 20, where only $P@20 = 0.38$ is obtained.

The results suggest that the TGR model is less effective in precision and recommendation classification than the established GNN-based models when a minimum number of nodes is not used. This difference could be due to several factors, such as the specific model architecture, the choice of hyperparameters, or the datasets used. It is important to note that the effectiveness of recommendation systems strongly depends on the characteristics of the application domain and the underlying data. Furthermore, a direct comparison between these models is not feasible since TGR is explicitly designed to allow internal comparison with the *Refurbed* data set and is therefore not optimized for small session sizes, as suggested and found beneficial in other studies [Wu+20a]. Future experiments should investigate whether the models used here for comparison perform better or equally well with a minimum number of nodes or how TGR performs with a completion strategy.

In general, the analysis of the results indicates that the TGR model needs to improve in terms of precision and classification accuracy compared to established GNN-based models. However, several approaches and possibilities exist to improve the model's performance and effectiveness as a recommendation system. Future research should focus on exploring and implementing such approaches to optimize the quality and relevance of user recommendations. It should be noted, however, that this comparison only considers positive metrics. Including a comparison using the metrics, $R@K^n$, would be beneficial as it can favor TGR or reveal the strengths and weaknesses of the other models.

5.6.8. Open topics

This section presents a list of challenges that warrant exploration in future work, particularly in the context of the *Diginetica* data set.

$MRR@K - P@K$ The observations indicate that when an item is hit in $P@K$, it is unlikely to have a position of 1. A similar trend is expected for the corresponding values of $MRR@K$ [Wu+20a; Gao+21]. These results may indicate possible inconsistencies in the model that require further investigation.

Criterion Alternative criteria that may better fit this model have not yet been evaluated. Research in this direction would be beneficial.

use-session-time-save Further empirical research is warranted to assess the suitability of this hyperparameter. In addition, possible alternative formulations for optimizing its value should be explored.

use-category-switch A comprehensive analysis is required to evaluate the suitability of this hyperparameter. Exploring other methods such as KG can also be beneficial.

Min/Max number of edges More research is needed to determine how many edges each node (user or item) requires in the bipartite graph so that this model does not lose its ability to generate recommendations. Similarly, more research is needed to determine if a maximum number of edges reduces the predictive quality, as the prediction of these is more likely than more unknown nodes.

embedding-dimension-for-user for use-dimension-for-user The optimal hidden size for this data set has not been adequately investigated. More research is needed to determine the most appropriate hidden size of this particular task.

In general, uncertainties remain regarding the suitability of this model and potential avenues for improvement. Thorough research is needed to determine the threshold at which the model's ability to detect and predict sessions effectively diminishes. Therefore, it should be conclusive whether this model is appropriate for the *Diginetica* data set.

5.7. Conclusion

In this thesis, several experiments have been conducted using 4 data sets. The results of these experiments are summarized and discussed in this section. The model used in this thesis TGR is defined in chapter 3. Here, the results for the data set *Diginetica* can be found in section 5.6, *Refurbed* in section 5.3 and *RetailRocket* in section 5.4.

Furthermore, this thesis is contextualized with the baseline models listed in section 5.2.

In Figure 5.27, a comparison of the model results is presented, focusing on different run modes. The results of the best runs per data set are shown and compared to the *CORE*, *SR-GNN*, and *Uni-GNN* models. The *CORE* model (section 5.2) has been selected as the baseline model with the best performance. *SR-GNN* (section 5.2) is chosen due to its similarity to the approach proposed in TGR-SES. Since *Refurbed* can only be compared to *Uni-GNN* (subsection 5.3.7), it is also included in the figure.

The best configurations for the *Refurbed* data set are run 4 of TGR-STR, run 10 of TGR-SES, and run 12 of TGR-HIS. For the *Diginetica* dataset, run 1 of TGR-STR, run 2 of TGR-SES, and run 13 of TGR-HIS were selected as the best configurations. For the data set *RetailRocket*, run 7 of TGR-STR, run 10 of TGR-SES, and run 13 of TGR-HIS are selected. Finally, for the data set *YooChoose*, run 1 of TGR-STR, run 10 of TGR-SES, and run 3 of TGR-HIS are selected.

The results indicate that TGR-HIS performs best with the configurations of runs 12 and 13, which implies that the usage of USE-CATEGORY-SWITCH and USE-SESSION-TIME-SAVE should be avoided. TGR-SES is most effective with the configurations of run 10, indicating that USE-CATEGORY-SWITCH should also be eliminated. Since the results of TGR-STR are not directly comparable to the others, further discussion of this run mode is omitted.

For the data set *Diginetica*, the experiments showed that TGR-HIS in combination with MOST-SIMILAR and USE-SESSION-TIME-SAVE $= 0$ produced the best results. This configuration achieved impressive values for $P@10 = MRR@10 = R@10^{100} = 0.94$, although the differences between the three metrics are minor and can be ignored. Additionally, the minimum sequence length per session was necessary for this data set because the

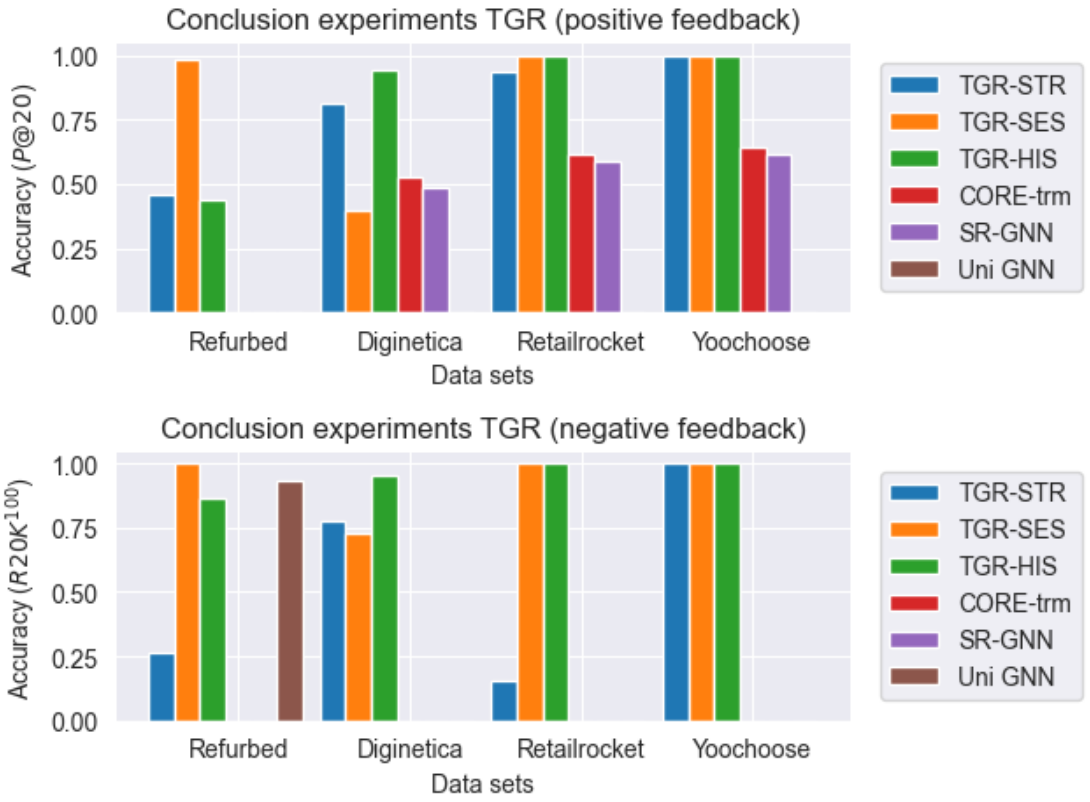


Figure 5.27.: Comparison of the best performing runs of TGR with a configuration parameter of $\mathcal{K} = 20$ with baseline models. Since these configurations have been the most successful ones selected for TGR, this analysis may have limitations in terms of comparability when compared to results from other models.

results with $P@10 = 0.3795$ are significantly worse since no minimum length was set.

For the data set *Refurbed*, TGR-SES in combination with MOST-SIMILAR and USE-CATEGORY-SWITCH $= 0$ resulted in the best results. The values obtained for $P@10 = MRR@10 = 0.9854$ and $R@10^{100} = 1$ are also close to perfection. In contrast to the data set *Diginetica*, this data set focuses specifically on items and users that are frequently used. Furthermore, in this data set, the information obtained by combining with sessions appears to be detrimental to the results.

For the data set *RetailRocket*, TGR-HIS combined with MOST-SIMILAR and USE-CATEGORY-SWITCH $= 0$ provided the most accurate recommendations. This setup achieved perfect values for $P@10 = MRR@10 = R@10^{100} = 1$, indicating a very accurate

identification of relevant recommendations. Interestingly, TGR-SES also appears to produce similar results. However, as in *Refurbed*, the use of USE-CATEGORY-SWITCH seems to have a negative impact on the results.

For the data set *YooChoose*, TGR-SES combined with RANDOM and USE-CATEGORY-SWITCH = 0 provided the most accurate recommendations. This setup achieved perfect values for $P@10 = MRR@10 = R@10^{100} = 1$, indicating a highly accurate identification of relevant recommendations. The exciting thing is that this data set achieved perfect results with RANDOM, which was not the case with other data sets.

Comparison of TGR with the baseline models shows that the presented model outperforms the other available models when a minimum session length is specified.

Thus, the model presented in this thesis can take advantage of temporal dependencies and the sequential nature of user behavior, leading to improved recommendation accuracy and the ability to capture long-term user preferences. Such capabilities can be considered as the reason why the baseline models have outperformed. This comparison emphasizes the superiority and potential of recommendation systems as a promising approach in the field.

The TGR recommendation system achieved excellent performance in all three data sets. The choice of negative sampling and run modes significantly impacted the metric values. Although the STRUCTURAL mode often struggled to exclude irrelevant elements, the run modes that processed the session graph yielded better values. The negative sampling method that shows the most promising results appears to be MOST-SIMILAR.

These observations highlight the potential of utilizing hierarchical structures to integrate sequential and session recommendation systems into a recommendation system that employs DGNN. They also provide insight into the factors that impact the performance of these systems. More research is needed to understand the underlying issues and determine the optimal configurations for specific use cases, which are discussed separately in section 6.2.

6. Conclusion

This chapter provides a comprehensive overview of the development of TGR, which combines session-based and sequence-based recommendation systems. It discusses the objectives achieved and suggests possibilities for future research to enhance the model. The summary of this thesis can be found in section 6.1. Furthermore, open research challenges that could not be addressed within the scope of this thesis are presented and categorized in section 6.2. Finally, section 6.3 provides an outlook on the significance of this research in the field of recommendation systems and suggests future research directions.

6.1. Summary

This thesis integrates session-based and sequential recommendation systems to generate personalized recommendations based on user behavior and context.Based on the results, this approach could increase user engagement and satisfaction [Guo20; E Ç17]. The TGR model uses DGNN to capture and model user behavior in multiple sessions to improve the effectiveness of combined recommendation systems. In TGR, the dynamic graph is updated during training to reflect the current structure and relationships in the graph. To ensure an accurate prediction, TGR consists of three layers.

The initial layer, TGR-STR, serves as the model's foundation and takes advantage of the ability DGNNs to learn dynamic information from neighboring nodes. It considers the neighborhood relations for each user $u \in \mathcal{U}$ and each item $i \in \mathcal{I}$ within a global bipartite graph. Based on this layer, TGR-SES focuses on processing user sessions as individual session graphs. Each session graph represents the interactions of a user within a session. These session graphs are processed using GNNs to extract relevant information for recommendations. The layer TGR-HIS integrates sequential and session-based recommendation systems into a unified model. It combines a user's historical session graphs with the current session to generate improved recommendations.

In summary, TGR provides relevant and personalized recommendations by learning

from user interactions. It uses graph-based methods and attention mechanisms to accommodate the dynamic relationships between users, articles, and interactions over time, resulting in personalized recommendations. This approach leads to enhanced personalization of recommendations, subsequently leading to increased user satisfaction. A comparison with several relevant baseline models in the research demonstrates the impressive dominance of TGR, especially when considering the minimal duration of the session. The results highlight the potential of the hierarchical structure of the TGR model to combine session-based and sequential recommendation systems with a DGNN.

The TGR recommendation system performed well in all four data sets. These results contribute to advancing recommendation system research and guide future extensions and optimizations.

Addressing the open issues identified in section 6.2 has considerable potential to significantly transform recommendation systems, making this approach a promising avenue for exploration. As outlined in section 1.2, these challenges are on the verge of being resolved conclusively. Successfully addressing these issues is expected to lead to more robust, efficient, and effective systems that better meet the needs of users and stakeholders and, therefore, should be investigated further.

6.2. Future Work

This section summarizes the challenges identified in this thesis, discussing the challenges, and proposing possible solutions.

Incorporating the neighborhood relationships of categories and users, as well as their social relationships, in the embedding of the layer TGR-STR can be beneficial, as current research by Zhou et al. [Zho+23] suggests. One main challenge is developing effective methods to model neighborhood relationships between items and categories. Capturing patterns in relationships between different items can help improve the accuracy of recommendations. By incorporating category neighborhoods, the model could identify similar items based on their category membership, providing more accurate recommendations [Zho+23]. However, implementing this feature presents some challenges, as the layer would need to be structurally modified to accommodate the additional information.

A significant research challenge related to GNN-based recommendation systems is investigating the effects of different combinations of hyperparameters. Although progress has been made in optimizing hyperparameters, unexplored combinations may still positively or negatively impact the model's accuracy. An overview of this topic can be found

in subsection 3.7.1.

It should be noted that the selection of optimal hyperparameters is highly dependent on the specific characteristics of the data set. Therefore, other data sets not studied in this thesis may require different parameters. Exploring which parameters are most appropriate for different data sets could be a subject of future research.

Several challenges emerged when evaluating the TGR model using the data sets *Diginetica*, *Refurbed*, *YooChoose*, and *RetailRocket*. When evaluating the accuracy of the recommendation using metrics $MRR@K$ and $P@K$, it was found that they are very similar and almost identical, resulting in $\delta < 0.1$. This indicates a problem in the model that the author could not identify. This problem could be significant and therefore requires further investigation in the future. To investigate whether there is a fundamental problem with the model, the edges of the user were replaced by random items in the *Diginetica* data set. Accordingly, $(u_1, i_1) \rightarrow (u_1, i_r)$ where r is a random number selected from all possible product indices. In this experiment, $P@K = MRR@K = 0.6813$ has been obtained. This corresponds to δ for the comparison experiment with the real $\delta = 0.2876$ data set. It can be concluded that TGR can detect patterns in the data. However, it should be noted that $P@K$ and $MRR@K$ are still too similar and that the score may be too high for a random data set. In summary, it cannot be ruled out that there is a fundamental problem, although the model has demonstrated some ability to predict patterns. Potential causes could be in the implementation of the metrics themselves. Another reason could be that the random comparison dataset based on the *Diginetica* dataset does not provide true randomness. Since this data set is relatively limited, certain patterns may have emerged. It is also possible that the scoring calculation is incorrect. There could be insufficient pre-processing of the data, including incorrectly coded data or other anomalies that could affect the evaluation of the model. Since the TGR model may be too complex, this could lead to overfitting. Furthermore, the optimization of the model may not have converged during training, leading to sub-optimal weights and poorer performance. Optimization was not examined during the experiments. However, the author found no errors, so these possible causes may not be complete or even the cause.

Another concern is the optimization criterion of GNN models. A more detailed investigation of the criterion and the exploration of possible adjustments could improve the performance of TGR, as recent research by Murata and Afzal [MA18] has shown.

Furthermore, there are exciting approaches to using a loss function other than the BCE Loss function, based on the findings of Raziperchikolaei and Chung [RC22]. Other func-

tions could significantly influence the predictions, but their impact was not investigated in this thesis and could be explored in future research [Li+23; RC22].

In addition, problems with session storage $\mathcal{M}^{\mathcal{S}}$ and processing are observed in TGR-HIS. Research indicates that disabling the manipulation of session embedding using the USE-SESSION-TIME-SAVE parameter can lead to degraded results, as observed in the *Diginetica* and *Refurbed* data sets. The process is described in subsection 3.4.2. This indicates that this model struggles to capture the temporal aspect of sessions accurately. Improving the modeling of session timing could lead to more accurate recommendations. In theory, the model should consider the timing of the session to recognize which session is more recent. Another challenge observed is the switching of categories. The model struggled to respond appropriately to change in categories during sessions, as demonstrated in experiments with the data sets *Diginetica* and *Refurbed*. As a result, the recommendations did not always align with the current interests and preferences of the users. Better consideration of category changes could improve the accuracy and relevance of recommendations. It is worth considering whether this could be accomplished by integrating it into the TGR-STR layer.

In addition, TGR potentially compromises scalability as the number of items per session decreases. Regulation of this characteristic is managed within the model by parameters MIN-COUNT-USER and MIN-COUNT-ITEM. This limitation could affect the adaptability of model applications in different contexts. A thorough investigation is warranted to determine whether this limitation is globally applicable or peculiar to the *Diginetica* data set. This assertion is significant, considering that the *RetailRocket* data set has no constraints. From a comprehensive analysis, it can be concluded that a more in-depth exploration of this topic is essential. In particular, this model excludes consideration of the *cold-start* problem. Although comparative models do not explicitly address this aspect, it can be inferred that its potential impact may have been considered during their development. As a result, a direct comparison of these models becomes complicated, as demonstrated by the *Diginetica* data set. However, this phenomenon does not manifest itself in the *RetailRocket* data set, where shorter sessions have no discernible effect. It should be possible to incorporate the *cold-start* problem into TGR, and this issue should be considered in future research efforts. The likely solutions to this challenge are described in subsection 2.3.1.

Furthermore, in the data sets *Diginetica* and *RetailRocket*, as shown in section 5.6 and section 5.4, the selected dimension of the user embeddings (EMBEDDING-DIMENSION-FOR-USER) may not be optimal for accurately representing user preferences. Inadequate

representation of user preferences can have a negative impact on the quality of recommendations. Careful adjustment of the embedding dimensions can address this issue and improve the model's performance. However, for the *RetailRocket* data set, this problem can only be identified theoretically, as near-perfect results were obtained with $P@K = 1$.

Interestingly, in the context of the current research, TGR seems to be optimally suited for session data, as evidenced by the experiments with the *YooChoose* data set. The data set *YooChoose* consists entirely of session data, making it particularly suitable for use with TGR. The data set *YooChoose* does not contain any item features. However, these two limitations do not have a significant negative impact. Experiments with the *YooChoose* data set indicate that SESSION provides the most accurate recommendations. Increased values for metrics $P@K$, $MRR@K$, and $R@K^n$ were recorded, indicating an accurate discrimination of the relevant recommendations.

Another important aspect that requires improvement is the computational efficiency of the model, particularly in terms of its training time. Although current training times have not been empirically measured, this component cannot be assumed to be optimal. The basis for this assumption is that per batch, TGR-SES and TGR-HIS are calculated independently using the historical session data. This indicates that batch processing, in its current form, is either not applicable or has not been implemented yet. In addition, using memory, such as $\mathcal{M}$, involves multiple memory accesses, which is likely to have a negative impact.

For this reason, optimizing the computational performance of the model should be a priority in subsequent stages of model development. Various strategies could be used to reduce training times, including algorithmic improvements, efficient data preprocessing methods, and distributed or parallel computing techniques. In addition, managing model memory can also be an important area for improvement. Reducing the frequency of memory accesses and optimizing memory usage could further accelerate the training process and enhance overall computational performance. Techniques such as caching, memory pooling, or using more efficient data structures could be explored in this context. The identified problems highlight the need for further investigation and improvements to optimize the performance and accuracy of the TGR model in the data sets mentioned. By conducting a thorough analysis and addressing these specific issues, the quality of recommendations can be significantly improved in the future, and the practical applicability of the model can be expanded.

6.3. Outlook

Despite the remarkable results obtained with TGR, extensive research is still required in recommendation systems.

One direction for future research is to increase the complexity of the model. While TGR already combines session-based and sequential recommendation systems, there is potential to integrate additional layers. In this approach, the incorporation of contextual information can be enhanced, resulting in more precise recommendations. By incorporating additional contextual factors such as categories or social connections, recommendations can be tailored to specific contexts, thus improving personalization, as discussed in section 6.2.

Another prominent area of research in recommendation systems is the integration of multimodal data [Zho+23]. Platforms operate on a variety of data, including text, images, audio, and video information. Future research could focus on integrating multimodal data into TGR to provide recommendations based on a more diverse database [Zho+23].

Diversity and explainability are major concerns in recommendation systems. However, these aspects have not been investigated in this thesis. Diversity, for instance, involves recommending items that are as diverse as possible to the user, thereby enabling exploration of the entire system. More details can be found in subsubsection 2.1.3. Although several metrics for measuring diversity already exist, they have not been tested in the current thesis [ABR19]. Another research topic is how TGR operates in the context of other platforms outside of e-commerce. This work has focused on the e-commerce domain, but TGR could also be used in other domains.

In addition, future research should focus on improving the scalability and efficiency of TGR. Recommendation systems require the efficient processing of large data sets and the management of numerous users and items. Therefore, it is crucial to further develop TGR and ensure its applicability in real-time scenarios with many users.

In summary, TGR provides a solid foundation for further developing future recommendation systems. The proposed research directions offer promising opportunities to improve personalization. The future of recommendation systems lies in the combination of different layers to create a deep representation of user preferences and context. TGR generates high-quality recommendations and improves the user experience. In summary, TGR achieves near-perfect accuracy in all metrics used in this thesis and in all data sets used for evaluation.

List of Abbreviations

AUGRU Attention Updating Gate. 38–40

BCE Loss Binary Cross Entropy Loss. 78, 181

CL4Rec Contrastive Learning for Sequential Recommendation. 121

CNN Convolutional Neural Network. 15

CORE Simple and Effective Session-Based Recommendation within Consistent Representation Space. 121

CTDG Continuous-time dynamic graph. 25

DAT-MDI Dual Attention Multi-Dimensional Integration. 36

DGNN Dynamic Graph Neural Network. iii, v, xv, 4, 6, 13, 15, 23–25, 28, 51, 52, 59, 60, 64, 177, 179, 180

DGTN Dual-channel Graph Transition Network. 36

DHCN Self-Supervised Hypergraph Convolutional Networks. 37

DTDG Discrete-time dynamic graph. 25

FPMC Factorized Personalized Markov Chains. 118

GAT Graph Attention Network. 13, 18, 42, 44

GC-SAN Graph Convolutional Self-Attention Network. 120, 121

GCE-GNN Global Context Enhanced Graph Neural Networks. 36

GCN Graph Convolutional Network. 13, 17, 42, 43, 120

GGNN Gated Graph Neural Network. 13, 19, 38

GGS-NN Gated Graph Sequence Neural Networks. 68, 82

GNN Graph Neural Network. xv, 2, 5, 6, 13, 15–22, 25–28, 33–35, 38–40, 42, 43, 45–47, 51, 68, 119, 120, 133, 136, 145, 147, 148, 158, 162, 166, 173, 174, 179–181

GRU Gated Recurrent Unit. 6, 19, 32, 38–40, 43, 64, 65, 118, 119

GRU4Rec Gated Recurrent Unit for Recommendation Systems. 118

HETGNN Heterogeneous Graph Neural Network. 19, 35

HGNN Hypergraph Neural Network. 20

KG Knowledge Graph. 10, 27, 134, 174

LESSR Graph-less Neural Networks. 36, 119

LSTM Long short-term memory. 38

MA-GNN Metapath Aggregated Graph Neural Network. 36

MATN Multi-Head Attention Network. 63–65

MLP Multilayer Perceptron. 53, 58, 59

NARM Neural Attentive Recommendation Model. 119

NISER+ Normalized Item and Session Representations to Handle Popularity Bias. 119

NN Neural Network. 1, 2, 5, 17, 22, 23, 26, 29, 31, 54, 58, 59, 61, 70, 133

PGNN Physics-guided Neural Networks. 36

RNN Recurrent Neural Network. 9, 19, 26, 32, 38, 53, 118, 173

RNS Random Negative Sampling. 61, 62

SASRec Self-Attentive Sequential Recommendation. 120

SBNS Similarity-based Negative Sampling. 62

SGNN-HN Star Graph Neural Networks. 36, 120

SHARE Session Recommendation with Hypergraph Attention Networks. 37

SR-GNN Session-based Recommendation with Graph Neural Networks. 37, 69, 119

TASRec Task Recommendation. 36

TGN Inductive representation learning on temporal graphs. 63

TGN Temporal Graph networks for deep Learning on dynamic Graphs. 52, 59, 60, 77

TGR Temporal Graph Recommendation. iii, v, xiii, xvi, 51, 53–55, 59–61, 66, 73, 76, 79–81, 83, 86, 90, 94, 98, 106–108, 113, 114, 117, 130, 133, 141, 145–149, 156–159, 169, 172–177, 179–184

TGR-HIS Temporal Graph Recommendation Historical Layer. 52, 53, 56, 71, 73, 75, 77, 125, 136, 141, 155, 162, 167, 169, 175, 176, 179, 182, 183

TGR-SES Temporal Graph Recommendation Session Layer. 52, 53, 56, 66, 67, 73, 77, 82, 125, 128, 138, 141, 152, 162, 164, 167, 169, 175–177, 179, 183

TGR-STR Temporal Graph Recommendation Structural Layer. 51–53, 56, 59, 60, 62, 66, 67, 71, 75–77, 125, 175, 179, 180, 182

Bibliography

[ABR19] A. Antikacioglu, T. Bajpai and R. Ravi. *A new system-wide diversity measure for recommendations with efficient algorithms*. 2019. arXiv: 1812.03030 [cs.IR].

[Arm+19] M. Armandpour, P. Ding, J. Huang and X. Hu. 'Robust Negative Sampling for Network Embedding'. In: *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019. DOI: 10.1609/aaai.v33i01.33013191.

[Ber+11] T. Bertin-Mahieux, D. Ellis, B. Whitman and P. Lamere. 'The Million Song Dataset'. In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*. 2011.

[Bi+20] Y. Bi, L. Song, M. Yao, Z. Wu, J. Wang and J. Xiao. *A Heterogeneous Information Network based Cross Domain Insurance Recommendation System for Cold Start Users*. 2020. arXiv: 2007.15293 [cs.IR].

[BS18] B. Bansal and S. Srivastava. 'Sentiment classification of online consumer reviews using word vector representations'. In: *Procedia Computer Science* 132 (2018). International Conference on Computational Intelligence and Data Science, pp. 1147–1153. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.05.029.

[BSF94] Y. Bengio, P. Simard and P. Frasconi. 'Learning long-term dependencies with gradient descent is difficult'. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[Bur02] R. Burke. 'Hybrid Recommender Systems: Survey and Experiments'. In: *User Modeling and User-Adapted Interaction* 12 (Nov. 2002). DOI: 10.1023/A:1021240730564.

[CGS21] C. Chen, J. Guo and B. Song. 'Dual Attention Transfer in Session-Based Recommendation with Multi-Dimensional Integration'. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 869–878. ISBN: 9781450380379. URL: https://doi.org/10.1145/3404835.3462866.

[Cha+20] B. Chang, G. Jang, S. Kim and J. Kang. 'Learning Graph-Based Geographical Latent Representation for Point-of-Interest Recommendation'. In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. CIKM '20. Virtual Event, Ireland: Association for Computing Machinery, 2020, pp. 135–144. DOI: 10.1145/33405\&31.3411905.

[Cha+21] J. Chang, C. Gao, Y. Zheng, Y. Hui, Y. Niu, Y. Song, D. Jin and Y. Li. *Sequential Recommendation with Graph Neural Networks*. 2021. DOI: 10.48550/ARXIV.2106.14226.

[Che+16] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu and H. Shah. *Wide and Deep Learning for Recommender Systems*. 2016. DOI: 10.48550/ARXIV.1606.07792.

[Chu+14] J. Chung, C. Gulcehre, K. Cho and Y. Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. arXiv: 1412.3555 [cs.NE].

[CW20] T. Chen and R. Wong. *Handling Information Loss of Graph Neural Networks for Session-based Recommendation*. Aug. 2020. DOI: 10.1145/3394486.3403170.

[CW21] T. Chen and R. Wong. 'An Efficient and Effective Framework for Session-Based Social Recommendation'. In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. WSDM '21. Virtual Event, Israel: Association for Computing Machinery, 2021, pp. 400–408. DOI: 10.1145/3437963.3441792.

[Das+21] T. Dash, S. Chitlangia, A. Ahuja and A. Srinivasan. *Incorporating Domain Knowledge into Deep Neural Networks*. 2021. arXiv: 2103.00180 [cs.NE].

[Daw+17] A. Daw, A. Karpatne, W. Watkins, J. Read and V. Kumar. *Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling*. 2017. DOI: 10.48550/ARXIV.1710.11431.

[DIG16] DIGINETICA. *CIKM Cup 2016 Track 2: Personalized E-Commerce Search Challenge*. 2016. URL: competitions.codalab.org/competitions/11161 (visited on 22/02/2023).

[DSC22] S. Dubey, S. Singh and B. Chaudhuri. *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*. 2022. arXiv: 2109.14545 [cs.LG].

[E Ç17] M. Morisio E. Ç ano. 'Hybrid recommender systems: A systematic literature review'. In: *Intelligent Data Analysis* 21.6 (Nov. 2017), pp. 1487–1524. DOI: 10.3233/ida-163209.

[Eln+21] A. Elngar, M. Arafa, A. Fathy, B. Moustafa, O. Mahmoud, M. Shaban and N. Fawzy. 'Image Classification Based On CNN: A Survey'. In: *Journal of Cybersecurity and Information Management* (Jan. 2021), PP. 18–50. DOI: 10.54216/JCIM.060102.

[Fan+21] Z. Fan, Z. Liu, J. Zhang, Y. Xiong, L. Zheng and P. Yu. *Continuous-Time Sequential Recommendation with Temporal Graph Collaborative Transformer*. 2021. DOI: 10.48550/ARXIV.2108.06625.

[FC20] M. Manzato F. de Aguiar Neto A. Costa and R. Campello. 'Pre-processing approaches for collaborative filtering based on hierarchical clustering'. In: *Information Sciences* 534 (2020), pp. 172–191. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2020.05.021.

[Fen+19] Y. Feng, F. Lv, W. Shen, M. Wang, F. Sun, Y. Zhu and K. Yang. *Deep Session Interest Network for Click-Through Rate Prediction*. 2019. arXiv: 1905.06482 [cs.IR].

[Gam17] J. Gamboa. *Deep Learning for Time-Series Analysis*. 2017. arXiv: 1701.01887 [cs.LG].

[Gao+21] C. Gao, Y. Zheng, N. Li, Y. Li, Y. Qin, J. Piao, Y. Quan, J. Chang, D. Jin, X. He and Y. Li. *Graph Neural Networks for Recommender Systems: Challenges, Methods, and Directions*. 2021. DOI: 10.48550/ARXIV.2109.12843.

[Guo20] Y. Guo. 'A Joint Neural Network for Session-Aware Recommendation'. In: *IEEE Access* 8 (2020), pp. 74205–74215. DOI: 10.1109/ACCESS.2020.2984287.

[Gup+21] P. Gupta, D. Garg, P. Malhotra, L. Vig and G. Shroff. *NISER: Normalized Item and Session Representations to Handle Popularity Bias*. 2021. arXiv: 1909.04276 [cs.IR].

[Hid+16] B. Hidasi, A. Karatzoglou, L. Baltrunas and D. Tikk. *Session-based Recommendations with Recurrent Neural Networks*. 2016. arXiv: 1511.06939 [cs.LG].

[Hid+18] B. Hidasi, M. Quadrana, A. Karatzoglou and D. Tikk. 'Parallel recurrent neural network architectures for feature-rich session-based recommendations'. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM. 2018, pp. 843–852.

[HL21] C. Hsu and C. Li. *RetaGNN: Relational Temporal Attentive Graph Neural Networks for Holistic Sequential Recommendation*. 2021. DOI: 10.48550/ARXIV.2101.12457.

[Hog+21] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. De Melo, C. Gutierrez, S. Kirrane, J. Gayo, R. Navigli, S. Neumaier, A. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab and A. Zimmermann. 'Knowledge Graphs'. In: *ACM Computing Surveys* 54.4 (July 2021), pp. 1–37. DOI: 10.1145/3447772.

[Hou+22] Yupeng Hou, Binbin Hu, Zhiqiang Zhang and Wayne Xin Zhao. *CORE: Simple and Effective Session-based Recommendation within Consistent Representation Space*. 2022. arXiv: 2204.11067 [cs.IR].

[Hu+19] L. Hu, S. Jian, L. Cao, Z. Gu, Q. Chen and A. Amirbekyan. 'HERS: Modeling Influential Contexts with Heterogeneous Relations for Sparse and Cold-Start Recommendation'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 3830–3837. DOI: 10.1609/aaai.v33i01.33013830.

[Hua+21] Z. Huang, D. Wu, Z. Weng, Y. Zhu and Z. Bai. 'Sequence-Aware Graph Neural Network for -based Recommendation'. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. 2021, pp. 1–8. DOI: 10.1109/IJCNN52387.2021.9533858.

[HY21] J. Huang and J. Yang. *UniGNN: a Unified Framework for Graph and Hypergraph Neural Networks*. 2021. arXiv: 2105.00956 [cs.LG].

[HYL17] W. Hamilton, R. Ying and J. Leskovec. *Inductive Representation Learning on Large Graphs*. 2017. DOI: 10.48550/ARXIV.1706.02216.

[JLJ15] D. Jannach, L. Lerche and M. Jugovac. 'Adaptation and Evaluation of Recommendations for Short-Term Shopping Goals'. In: *Proceedings of the 9th ACM Conference on Recommender Systems*. RecSys '15. Vienna, Austria: Association for Computing Machinery, 2015, pp. 211–218. DOI: 10.1145/2792838.2800176.

[Jun22] A. Jung. *Machine Learning: The Basics*. 2022. arXiv: 1805.05052 [cs.LG].

[Kaz+19] S. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth and P. Poupart. *Representation Learning for Dynamic Graphs: A Survey*. 2019. DOI: 10.48550/ARXIV.1905.11485.

[KM18] W. Kang and J. McAuley. *Self-Attentive Sequential Recommendation*. 2018. arXiv: 1808.09781 [cs.IR].

[KML18] K. Kang, J. McAuley and J. Leskovec. 'Self-attentive sequential recommendation'. In: *Proceedings of the 2018 IEEE International Conference on Data Mining*. IEEE. 2018, pp. 197–206.

[Kum23] L. Kummer. *Lorenz Kummer, BSc MSc*. https://ufind.univie.ac.at/de/person.html?id=104353. access 05.06.2023. 2023.

[Li+15] Y. Li, D. Tarlow, M. Brockschmidt and R. Zemel. *Gated Graph Sequence Neural Networks*. 2015. DOI: 10.48550/ARXIV.1511.05493.

[Li+17] J. Li, P. Ren, Z. Chen, Z. Ren and J. Ma. *Neural Attentive Session-based Recommendation*. 2017. arXiv: 1711.04725 [cs.IR].

[Li+20] Z. Li, X. Zhu, Y. Song and C. Li. 'Session-based Recommendation with Graph Convolutional Networks'. In: *IEEE Transactions on Knowledge and Data Engineering* 32.12 (2020), pp. 2449–2460.

[Li+21a] W. Li, M. He, Z. Huang, X. Wang, S. Feng, W. Su and Y. Sun. *Graph4Rec: A Universal Toolkit with Graph Neural Networks for Recommender Systems*. 2021. arXiv: 2112.01035 [cs.LG].

[Li+21b] Y. Li, T. Chen, Y. Luo, H. Yin and Z. Huang. *Discovering Collaborative Signals for Next POI Recommendation with Iterative Seq2Graph Augmentation*. 2021. DOI: 10.48550/ARXIV.2106.15814.

[Li+23] F. Li, S. Yu, F. Zeng and F. Yang. *Effective and Efficient Training for Sequential Recommendation Using Cumulative Cross-Entropy Loss*. 2023. arXiv: 2301.00979 [cs.IR].

[Lim+20] N. Lim, B. Hooi, S. Ng, X. Wang, Y. Goh, R. Weng and J. Varadarajan. 'STP-UDGAT: Spatial-Temporal-Preference User Dimensional Graph Attention Network for Next POI Recommendation'. In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. ACM, Oct. 2020. DOI: 10.1145/3340531.3411876.

[Liu+18] Q. Liu, Y. Zeng, R. Mokhosi and H. Zhang. 'STAMP: Short-Term Attention/Memory Priority Model for Session-Based Recommendation'. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 1831–1839. DOI: 10.1145/3219819.3219950.

[Liu+20] F. Liu, W. Liu, X. Li and Y. Ye. *Inter-sequence Enhanced Framework for Personalized Sequential Recommendation*. 2020. DOI: 10.48550/ARXIV.2004.12118.

[Liu+21] X. Liu, Z. Zhao, H. Ma, M. Wang and Y. Yang. 'Deep Dynamic Graph Neural Networks for Session-based Recommendation'. In: *IEEE Transactions on Knowledge and Data Engineering* 33.1 (2021), pp. 222–235.

[Liu+23] C. Liu, Y. Zhan, J. Wu, C. Li, C. Du, W. Hu, T. Liu and D. Tao. *Graph Pooling for Graph Neural Networks: Progress, Challenges, and Opportunities*. 2023. arXiv: 2204.07321 [cs.LG].

[Lv+19] F. Lv, T. Jin, C. Yu, F. Sun, Q. Lin, K. Yang and W. Ng. *SDM: Sequential Deep Matching Model for Online Large-scale Recommender System*. 2019. arXiv: 1909.00385 [cs.IR].

[Lyu+20] X. Lyu, G. Li, J. Huang and W. Hu. *Rule-Guided Graph Neural Networks for Recommender Systems*. 2020. arXiv: 2009.04104 [cs.LG].

[Ma+19] C. Ma, L. Ma, Y. Zhang, J. Sun, X. Liu and M. Coates. *Memory Augmented Graph Neural Networks for Sequential Recommendation*. 2019. DOI: 10.48550/ARXIV.1912.11730.

[MA18] T. Murata and N. Afzal. 'Modularity Optimization as a Training Criterion for Graph Neural Networks'. In: *Complex Networks IX*. Springer International Publishing, 2018, pp. 123–135. DOI: 10.1007/978-3-319-73198-8_11.

[MC92] M. Mavrovouniotis and S. Chang. 'Hierarchical neural networks'. In: *Computers and Chemical Engineering* 16.4 (1992). Neutral network applications in chemical engineering, pp. 347–369. ISSN: 0098-1354. DOI: https://doi.org/10.1016/0098-1354(92)80053-C.

[McA14] J. McAuley. *Amazon Product Data*. 2014. URL: http://jmcauley.ucsd.edu/data/amazon/links.html (visited on 22/02/2023).

[Meh+18] R. Mehrotra, J. McInerney, H. Bouchard, M. Lalmas and F. Diaz. 'Towards a Fair Marketplace: Counterfactual Evaluation of the Trade-off between Relevance, Fairness and Satisfaction in Recommendation Systems'. In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. CIKM '18. Torino, Italy: Association for Computing Machinery, 2018, pp. 2243–2251. DOI: 10.1145/3269206.3272027.

[Mik+13] T. Mikolov, K. Chen, G. Corrado and J. Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. DOI: 10.48550/ARXIV.1301.3781.

[Mit+22] S. Mitheran, A. Java, S. Sahu and A. Shaikh. *Introducing Self-Attention to Target Attentive Graph Neural Networks*. 2022. arXiv: 2107.01516 [cs.IR].

[Moc+16] M. Moczulski, M. Denil, J. Appleyard and N. Freitas. *ACDC: A Structured Efficient Linear Layer*. 2016. arXiv: 1511.05946 [cs.LG].

[MYX20] W. Meng, D. Yang and Y. Xiao. 'Incorporating User Micro-behaviors and Item Knowledge into Multi-task Learning for Session-based Recommendation'. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, July 2020. DOI: 10.1145/3397271.3401098.

[Pan+20] Z. Pan, F. Cai, W. Chen, H. Chen and M. de Rijke. 'Star Graph Neural Networks for Session-Based Recommendation'. In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. CIKM '20. Virtual Event, Ireland: Association for Computing Machinery, 2020, pp. 1195–1204. DOI: 10.1145/3340531.3412014.

[Par+19] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, T. Schardl and C. Leiserson. *EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs*. 2019. arXiv: 1902.10191 [cs.LG].

[Phu19] T. Phuong. 'Neural Session-Aware Recommendation'. In: *IEEE Access* 7 (2019), pp. 86884–86896. DOI: 10.1109/ACCESS.2019.2926074.

[QCJ18] M. Quadrana, P. Cremonesi and D. Jannach. *Sequence-Aware Recommender Systems*. 2018. DOI: 10.48550/ARXIV.1802.08452.

[Qiu+19] R. Qiu, J. Li, Z. Huang and H. YIn. 'Rethinking the Item Order in Session-based Recommendation with Graph Neural Networks'. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM, Nov. 2019. DOI: 10.1145/3357384.3358010.

[Qiu+20] R. Qiu, H. Yin, Z. Huang and T. Chen. 'GAG'. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, July 2020. DOI: 10.1145/3397271.3401109.

[QKH17] M. Quadrana, A. Karatzoglou and B. Hidasi. 'Personalizing session-based recommendations with hierarchical recurrent neural networks'. In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM. 2017, pp. 130–137.

[RC22] R. Raziperchikolaei and Y. Chung. *One-class Recommendation Systems with the Hinge Pairwise Distance Loss and Orthogonal Representations*. 2022. arXiv: 2208.14594 [cs.IR].

[Ren+12] S. Rendle, C. Freudenthaler, Z. Gantner and L. Schmidt-Thieme. *BPR: Bayesian Personalized Ranking from Implicit Feedback*. 2012. arXiv: 1205.2618 [cs.IR].

[Ret22] Retailrocket. *Retailrocket (www.kaggle.com/datasets/retailrocket)*. 2022. URL: www.kaggle.com/datasets/retailrocket.

[RFS10] S. Rendle, C. Freudenthaler and L. Schmidt-Thieme. 'Factorizing Personalized Markov Chains for Next-Basket Recommendation'. In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 811–820. DOI: 10.1145/1772690.1772773.

[Ros+20] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti and M. Bronstein. *Temporal Graph Networks for Deep Learning on Dynamic Graphs*. 2020. DOI: 10.48550/ARXIV.2006.10637.

[Ros21] E. Rossi. *TGN: Temporal Graph Networks for Dynamic Graphs*. 2021. URL: https://emanuelerossi.co.uk/assets/pdf/TGN%5C%5F2021%5C%5F01%5C%5F22.pdf (visited on 09/07/2023).

[Sch+22] T. Schnake, O. Eberle, J. Lederer, S. Nakajima, K. Schutt, K. Muller and G. Montavon. 'Higher-Order Explanations of Graph Neural Networks via Relevant Walks'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.11 (Nov. 2022), pp. 7581–7596. DOI: 10.1109/tpami.2021.3115452.

[SJS06] M. Sokolova, N. Japkowicz and S. Szpakowicz. 'Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation'. In: *F-Score and ROC*. Vol. Vol. 4304. Jan. 2006, pp. 1015–1021. DOI: 10.1007/11941439_114.

[SL20] H. Sheu and S. Li. 'Context-Aware Graph Embedding for Session-Based News Recommendation'. In: *Fourteenth ACM Conference on Recommender Systems*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 657–662. ISBN: 9781450375832. URL: https://doi.org/10.1145/3383313.3418477.

[Sob21] S. Sobolevsky. *Hierarchical Graph Neural Networks*. 2021. arXiv: 2105.03388 [cs.LG].

[Son+21] W. Song, S. Wang, Y. Wang and S. Wang. *Next-item Recommendations in Short Sessions*. 2021. arXiv: 2107.07453 [cs.IR].

[Sri+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting'. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[SS21] M. Santana and A. Soares. *Hybrid Model with Time Modeling for Sequential Recommender Systems*. 2021. arXiv: 2103.06138 [cs.IR].

[TOS20] R. Togashi, M. Otani and S. Satoh. *Alleviating Cold-Start Problems in Recommendation through Pseudo-Labelling over Knowledge Graph*. 2020. arXiv: 2011.05061 [cs.IR].

[Vas+17] A. Vaswan, N. Shazee, N. Parma, J. Uszkorei, L. Jone, A. Gomez, L. Kaiser and I. Polosukhin. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762.

[Vel+17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio and Y. Bengio. *Graph Attention Networks*. 2017. DOI: 10.48550/ARXIV.1710.10903.

[VF05] M. Verleysen and D. François. 'The Curse of Dimensionality in Data Mining and Time Series Prediction'. In: *Computational Intelligence and Bioinspired Systems*. Ed. by A. Prieto J. Cabestany and F. Sandoval. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 758–770. ISBN: 978-3-540-32106-4.

[Wan+19] H. Wang, F. Zhang, M. Zhang, J. Leskovec, M. Zhao, W. Li and Z. Wang. *Knowledge-aware Graph Neural Networks with Label Smoothness Regularization for Recommender Systems*. 2019. arXiv: 1905.04413 [cs.LG].

[Wan+20a] W. Wang, W. Zhang, S. Liu, Q. Liu, B. Zhang, L. Lin and H. Zha. *Beyond Clicks: Modeling Multi-Relational Item Graph for Session-Based Target Behavior Prediction*. 2020. DOI: 10.48550/ARXIV.2002.07993.

[Wan+20b] Z. Wang, W. Wei, G. Cong, X. Li, X. Mao and M. Qiu. 'Global Context Enhanced Graph Neural Networks for Session-based Recommendation'. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, July 2020. DOI: 10.1145/3397271.3401142.

[Wan+21a] J. Wang, K. Ding, Z. Zhu and J. Caverlee. 'Session-based Recommendation with Hypergraph Attention Networks'. In: *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. Society for Industrial and Applied Mathematics, Jan. 2021, pp. 82–90. DOI: 10.1137/1.9781611976700.10.

[Wan+21b] S. Wang, L. Hu, Y. Wang, X. He, Q. Sheng, M. Orgun, L. Cao, F. Ricci and P. Yu. *Graph Learning based Recommender Systems: A Review*. 2021. DOI: 10.48550/ARXIV.2105.06339.

[Wu+19] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie and T. Tan. 'Session-Based Recommendation with Graph Neural Networks'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (July 2019), pp. 346–353. DOI: 10.1609/aaai.v33i01.3301346.

[Wu+20a] W. Wu, F. Sun, W. Zhang, X. Xie and B. Cui. *Graph Neural Networks in Recommender Systems: A Survey*. 2020. DOI: 10.48550/ARXIV.2011.02260.

[Wu+20b] Z. Wu, W. Hu, Y. Zhang and Y. Liu. 'Session-based Recommendation with Graph Neural Networks'. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 2020, pp. 1293–1296.

[Wu+22] L. Wu, P. Cui, J. Pei and L. Zhao. *Graph Neural Networks: Foundations and Frontiers and Applications*. Singapore: Springer Singapore, 2022, p. 725.

[Xia+20] X. Xia, H. Yin, J. Yu, Q. Wang, L. Cui and X. Zhang. *Self-Supervised Hypergraph Convolutional Networks for Session-based Recommendation*. 2020. DOI: 10.48550/ARXIV.2012.06852.

[Xia+21] X. Xia, H. Yin, J. Yu, Y. Shao and L. Cui. *Self-Supervised Graph Co-Training for Session-based Recommendation*. 2021. DOI: 10.48550/ARXIV.2108.10560.

[Xie+16] M. Xie, H. Yin, F. Xu, H. Wang and X. Zhou. 'Graph-Based Metric Embedding for Next POI Recommendation'. In: *Proceedings of the 17th International Conference on Web Information Systems Engineering - Volume 10042*. WISE 2016. Berlin, Heidelberg: Springer-Verlag, 2016, pp. 207–222. DOI: 10.1007/978-3-319-48743-4_17.

[Xie+21a] X. Xie, Z. Liu, S. Wu, F. Sun, C. Liu, J. Chen, J. Gao, B. Cui and B. Ding. *CausCF: Causal Collaborative Filtering for RecommendationEffect Estimation*. 2021. DOI: 10.48550/ARXIV.2105.13881.

[Xie+21b] X. Xie, F. Sun, Z. Liu, S. Wu, J. Gao, B. Ding and B. Cui. *Contrastive Learning for Sequential Recommendation*. 2021. arXiv: 2010.14395 [cs.IR].

[Xu+19] C. Xu, P. Zhao, Y. Liu, V. Sheng, J. Xu, F. Zhuang and J. Fang. 'Graph Contextualized Self-Attention Network for Session-based Recommendation'. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. IJCAI'19. Macao, China: International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 3940–3946. DOI: 10.24963/ijcai.2019/547.

[Xu+20a] D. Xu, C. Ruan, E. Korpeoglu, S. Kumar and K. Achan. 'Inductive representation learning on temporal graphs'. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=rJeW1yHYwH.

[Xu+20b] Y. Xu, J. Chen, C. Huang, B. Zhang, H. Xing, P. Dai and L. Bo. 'Joint Modeling of Local and Global Behavior Dynamics for Session-Based Recommendation'. In: *ECAI*. 2020.

[Yan+20] Z. Yang, M. Ding, C. Zhou, H. Yang, J. Zhou and J. Tang. *Understanding Negative Sampling in Graph Representation Learning*. 2020. arXiv: 2005.09863 [cs.LG].

[Yao+22] N. Yao, Q. Liu, X. Li, Y. Yang and Q. Bai. 'Entity Similarity-Based Negative Sampling for Knowledge Graph Embedding'. In: *PRICAI 2022: Trends in Artificial Intelligence*. Ed. by Q. Bai S. Khanna J. Cao and G. Xu. Cham: Springer Nature Switzerland, 2022, pp. 73–87. ISBN: 978-3-031-20865-2.

[Yin+18] R. Ying, R. He, K. Chen, P. Eksombatchai, W. Hamilton and J. Leskovec. 'Graph Convolutional Neural Networks for Web-Scale Recommender Systems'. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, July 2018. DOI: 10.1145/3219819.3219890.

[Yoo22] Yoochoose. *Yoochoose*. 13th Aug. 2022. URL: https://ktakanov.github.io/datascience/2020/08/13/eda-recsys.html.

[You+20] J. You, J. Leskovec, K. He and S. Xie. *Graph Structure of Neural Networks*. 2020. DOI: 10.48550/ARXIV.2007.06559.

[Yu+20] F. Yu, Y. Zhu, Q. Liu, S. Wu, L. Wang and T. Tan. 'TAGNN: Target Attentive Graph Neural Networks for Session-based Recommendation'. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research*

and Development in Information Retrieval. ACM, July 2020. DOI: 10.1145/3397271.3401319.

[ZC20] Y. Zhang and X. Chen. 'Explainable Recommendation: A Survey and New Perspectives'. In: *Foundations and Trends® in Information Retrieval* 14.1 (2020), pp. 1–101. ISSN: 1554-0669. DOI: 10.1561/1500000066.

[Zha+19a] C. Zhang, D. Song, C. Huang, A. Swami and N. Chawla. 'Heterogeneous Graph Neural Network'. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 793–803. DOI: 10.1145/3292500.3330961.

[Zha+19b] S. Zhang, H. Tong, J. Xu and R. Maciejewski. 'Graph convolutional networks: a comprehensive review'. In: *Computational Social Networks* 6 (Nov. 2019). DOI: 10.1186/s40649-019-0069-y.

[Zha+21a] M. Zhang, S. Wu, M. Gao, X. Jiang, K. Xu and L. Wang. 'Personalized Graph Neural Networks with Attention Mechanism for Session-Aware Recommendation'. In: *IEEE Transactions on Knowledge and Data Engineering* (2021), pp. 1–1. DOI: 10.1109/tkde.2020.3031329.

[Zha+21b] M. Zhang, S. Wu, X. Yu, Q. Liu and L. Wang. *Dynamic Graph Neural Networks for Sequential Recommendation*. 2021. DOI: 10.48550/ARXIV.2104.07368.

[Zha21] X. Zhang. 'Dual Part-pooling Attentive Networks for Session-based Recommendation'. In: *Neurocomputing* 440 (2021), pp. 89–100. DOI: 10.1016/J.NEUCOM.2021.01.092.

[Zhe+20] Y. Zheng, S. Liu, Z. Li and S. Wu. *DGTN: Dual-channel Graph Transition Network for Session-based Recommendation*. 2020. DOI: 10.48550/ARXIV.2009.10002.

[Zhe+21] Y. Zheng, C. Gao, L. Chen, D. Jin and Y. Li. 'DGCN: Diversified Recommendation with Graph Convolutional Networks'. In: *Proceedings of the Web Conference 2021*. ACM, Apr. 2021. DOI: 10.1145/3442381.3449835.

[Zho+18] G. Zhou, N. Mou, Y. Fan, W. Pi, W. Bian and C. Zhou. 'Deep interest evolution network for click-through rate prediction'. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2018, pp. 1059–1068.

[Zho+21] H. Zhou, Q. Tan, X. Huang, K. Zhou and X. Wang. 'Temporal Augmented Graph Neural Networks for Session-Based Recommendations'. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1798–1802. ISBN: 9781450380379. URL: https://doi.org/10.1145/3404835.3463112.

[Zho+23] H. Zhou, X. Zhou, Z. Zeng, L. Zhang and Z. Shen. *A Comprehensive Survey on Multimodal Recommender Systems: Taxonomy, Evaluation, and Future Directions*. 2023. arXiv: 2302.04473 [cs.IR].

[Zhu+22] Y. Zhu, F. Lyu, C. Hu, X. Chen and X. Liu. *Encoder-Decoder Architecture for Supervised Dynamic Graph Learning: A Survey*. 2022. DOI: 10.48550/ARXIV.2203.10480.

[ZSC21] T. Zhu, L. Sun and G. Chen. 'Graph-based Embedding Smoothing for Sequential Recommendation'. In: *IEEE Transactions on Knowledge and Data Engineering* (2021), pp. 1–1. DOI: 10.1109/TKDE.2021.3073411.

[ZWL20] H. Zhang, X. Wu and X. Li. 'Dynamic Graph Convolutional Networks for Session-based Recommendation'. In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. ACM. 2020, pp. 2673–2676.

A. Appendix

A.1. Refurbed

In Table A.1 the results and the experimental setup of experiment A are shown for the data set *Refurbed* with $\mathcal{K} = 10$. Similarly, Table A.2 represents experiment B, Table A.3 represents experiment C, Table A.4 represents experiment D, and Table A.1 represents experiment E. In these tables, n_{hit} indicates whether the score of the target item t is within the first $\mathcal{K}$, which can be described by $y_t > y_{\mathcal{K}}$. Where y_t is the target item score, $\frac{y_t}{|X_{\text{test}}|}$ is the mean of all scores within the test data set. Furthermore, $\frac{y_{\mathcal{K}}}{|X_{\text{test}}|}$ represents the average of the highest values of $\mathcal{K}$ and $\frac{y}{|X_{\text{test}}|}$ describes the average in general. In addition, $\frac{\text{pos}}{|X_{\text{test}}|}$ reflects the average position within y. Also, LESS stands for LESS-SIMILAR and MOST stands for MOST-SIMILAR. The other points are as defined by subsection 2.5.2.

Hyperparamter	1	2	3
Experiment	A/B	A/B	A/B
MIN-COUNT-USER	5	5	5
MIN-COUNT-ITEM	5	5	5
REMOVE-COMMON-ITEMS	0.05	0.05	0.05
REMOVE-COMMON-USERS	0.05	0.05	0.05
RUN-MODE	STRUCTURAL	SESSION	HISTORY
NEGATIVE-SAMPLER	MOST	MOST	MOST
USE-CATEGORY-SWITCH	1	1	1
USE-SESSION-TIME-SAVE	1	1	1
Use MinMax	False	False	False
Top $\mathcal{K}$	10	10	10
Found	16975	39416	21046
Length	41800	41800	41800
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.5418	0.9291	0.6014
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.6154	0.5839	0.6159
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.4875	0.5079	0.5117
Pos	7535	903	5794
$P@K$	0.4061	0.9429	0.5034
$MRR@K$	0.3864	0.942	0.487
$R@\mathcal{K}^n$	0.0832	0.8552	0.0
$P@\mathcal{K}^n$	0.0008	0.0084	0.0
$F_1@\mathcal{K}^n$	0.0016	0.0166	0.0

Table A.1.: Results and setup of experiment A for the data set *Refurbed* with $\mathcal{K} = 10$

Hyperparameter	1	2	3	4	5	6	7	8	9
Experiment	A/B	A/B	A/B	B	B	B	B	B	B
MIN-COUNT-USER	5	5	5	5	5	5	5	5	5
MIN-COUNT-ITEM	5	5	5	5	5	5	5	5	5
REMOVE-COMMON-ITEMS	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
REMOVE-COMMON-USERS	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
RUN-MODE	STR	SES	HIS	STR	SES	HIS	STR	SES	HIS
NEGATIVE-SAMPLER	MOST	MOST	MOST	RANDOM	RANDOM	RANDOM	LESS	LESS	LESS
USE-CATEGORY-SWITCH	1	1	1	1	1	1	1	1	1
USE-SESSION-TIME-SAVE	1	1	1	1	1	1	1	1	1
Use MinMax	False	False	False	False	False	False	False	False	False
Top $\mathcal{K}$	10	10	10	10	10	10	10	10	10
Found	16975	39416	21046	19014	9608	6074	5787	41275	20822
Length	41800	41800	41800	41800	41800	41800	41800	41800	41800
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.5418	0.9291	0.6014	0.5448	0.4986	0.4948	0.4994	0.9166	0.5546
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.6154	0.5839	0.6159	0.6115	0.5758	0.6111	0.6094	0.571	0.6093
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.4875	0.5079	0.5117	0.4826	0.4915	0.5041	0.4721	0.4832	0.5071
Pos	7535	903	5794	7357	10007	10015	8557	132	7689
$P@K$	0.4061	0.9429	0.5034	0.4548	0.2298	0.1453	0.1384	0.6416	0.4981
$MRR@K$	0.3864	0.942	0.487	0.439	0.2128	0.1203	0.1145	0.9869	0.4926
$R@\mathcal{K}^n$	0.0832	0.8552	0.0	0.1488	0.0	0.2921	0.1527	1.0	0.0
$P@\mathcal{K}^n$	0.0008	0.0084	0.0	0.0014	0.0	0.0028	0.0029	0.0099	0.0
$F_1@\mathcal{K}^n$	0.0016	0.0166	0.0	0.0029	0.0	0.0057	0.0015	0.0196	0.0

Table A.2.: Results and setup of experiment B for the data set *Refurbed* with $\mathcal{K} = 10$. STR stands for STRUCTURAL, SES stands for SESSION and HIS stands for HISTORY.

Hyperparameter	10
Experiment	C
MIN-COUNT-USER	5
MIN-COUNT-ITEM	5
REMOVE-COMMON-ITEMS	0.05
REMOVE-COMMON-USERS	0.05
RUN-MODE	SESSION
NEGATIVE-SAMPLER	MOST
USE-CATEGORY-SWITCH	0
USE-SESSION-TIME-SAVE	1
Use MinMax	False
Top $\mathcal{K}$	10
Found	41190
Length	41800
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.8404
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.5542
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.5033
Pos	155
$P@K$	0.9854
$MRR@K$	0.9847
$R@\mathcal{K}^n$	1.0
$P@\mathcal{K}^n$	0.0099
$F_1@\mathcal{K}^n$	0.0196

Table A.3.: Results and setup of experiment D for the data set *Refurbed* with $\mathcal{K} = 10$

Hyperparameter	11	12	13	14
Experiment	D	D	D	D
MIN-COUNT-USER	5	5	5	5
MIN-COUNT-ITEM	5	5	5	5
REMOVE-COMMON-ITEMS	0.05	0.05	0.05	0.05
REMOVE-COMMON-USERS	0.05	0.05	0.05	0.05
RUN-MODE	HISTORY	HISTORY	HISTORY	HISTORY
NEGATIVE-SAMPLER	MOST	MOST	MOST	LESS
USE-CATEGORY-SWITCH	0	0	1	0
USE-SESSION-TIME-SAVE	0	0	0	0
Use MinMax	False	True	False	False
Top $\mathcal{K}$	10	10	10	10
Found	19001	17849	21217	16186
Length	41800	41800	41800	41800
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.5495	0.5244	0.5751	0.5507
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.6029	0.5725	0.5925	0.6188
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.5046	0.4277	0.4787	0.4979
Pos	7719	5741	5990	7820
$P@K$	0.4545	0.427	0.5075	0.3872
$MRR@K$	0.4521	0.4088	0.5017	0.3827
$R@\mathcal{K}^n$	0.2524	0.8629	0.8684	0.1207
$P@\mathcal{K}^n$	0.0021	0.0085	0.0085	0.0011
$F_1@\mathcal{K}^n$	0.0042	0.0169	0.017	0.0023

Table A.4.: Results and setup of experiment D for the data set *Refurbed* with $\mathcal{K} = 10$

Hyperparameter	15	16
Experiment	E	E
MIN-COUNT-USER	5	3
MIN-COUNT-ITEM	5	3
REMOVE-COMMON-ITEMS	0.1	0.05
REMOVE-COMMON-USERS	0.1	0.05
RUN-MODE	SESSION	SESSION
NEGATIVE-SAMPLER	MOST	MOST
USE-CATEGORY-SWITCH	0	0
USE-SESSION-TIME-SAVE	1	1
Use MinMax	False	False
Top $\mathcal{K}$	10	10
Found	9484	51270
Length	15600	53000
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.6528	0.9579
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.5866	0.5935
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.4823	0.5281
Pos	7973	513
$P@K$	0.6079	0.9673
$MRR@K$	0.6079	0.9668
$R@\mathcal{K}^n$	0.191	1.0
$P@\mathcal{K}^n$	0.0018	0.0099
$F_1@\mathcal{K}^n$	0.0037	0.0196

Table A.5.: Results and setup of experiment E for the data set *Refurbed* with $\mathcal{K} = 10$

A.2. Diginetica

In Table A.6 the results and the experimental setup of experiment A are shown for the data set *Diginetica* with $\mathcal{K} = 10$. Similarly, Table A.7 represents experiment B, Table A.8 represents experiment C, Table A.9 represents experiment D, and Table A.6 represents experiment E. In these tables, n_{hit} indicates whether the score of the target item t is within the first $\mathcal{K}$, which can be described by $y_t > y_{\mathcal{K}}$. Where y_t is the target item score, $\frac{y_t}{|X_{\text{test}}|}$ is the mean of all scores within the test data set. Furthermore, $\frac{y_{\mathcal{K}}}{|X_{\text{test}}|}$ represents the average of the highest values of $\mathcal{K}$ and $\frac{y}{|X_{\text{test}}|}$ describes the average in general. In addition, $\frac{\text{pos}}{|X_{\text{test}}|}$ reflects the average position within y. Also, LESS stands for LESS-SIMILAR and MOST stands for MOST-SIMILAR. The other points are as defined by subsection 2.5.2.

Hyperparameter	1	2	3
Experiment	A/B	A/B	A/B
MIN-COUNT-USER	5	5	5
MIN-COUNT-ITEM	5	5	5
REMOVE-COMMON-ITEMS	0.05	0.05	0.05
REMOVE-COMMON-USERS	0.05	0.05	0.05
RUN-MODE	STRUCTURAL	SESSION	HISTORY
NEGATIVE-SAMPLER	MOST	MOST	MOST
USE-CATEGORY-SWITCH	1	1	1
USE-SESSION-TIME-SAVE	1	1	1
Use MinMax	False	False	False
Top $\mathcal{K}$	10	10	10
Found	3103	1520	3782
Length	3800	3800	3800
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.65	0.5044	0.9629
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.5755	0.5131	0.5397
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.4949	0.4988	0.499
Pos	30399	77816	5
$P@K$	0.8165	0.4	0.9952
$MRR@K$	0.8165	0.4	0.9952
$R@\mathcal{K}^n$	0.0878	0.7289	0.5176
$P@\mathcal{K}^n$	0.0008	0.0072	0.0051
$F_1@\mathcal{K}^n$	0.0017	0.0142	0.0101

Table A.6.: Results and setup of experiment A for the data set *Diginetica* with $\mathcal{K} = 10$

Hyperparameter	1	2	3	4	5	6	7	8	9
Experiment	A/B	A/B	A/B	B	B	B	B	B	B
MIN-COUNT-USER	5	5	5	5	5	5	5	5	5
MIN-COUNT-ITEM	5	5	5	5	5	5	5	5	5
REMOVE-COMMON-ITEMS	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
REMOVE-COMMON-USERS	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
RUN-MODE	STR	SES	HIS	STR	SES	HIS	STR	SES	HIS
NEGATIVE-SAMPLER	MOST	MOST	MOST	RANDOM	RANDOM	RANDOM	LESS	LESS	LESS
USE-CATEGORY-SWITCH	1	1	1	1	1	1	1	1	1
USE-SESSION-TIME-SAVE	1	1	1	1	1	1	1	1	1
Use MinMax	False	False	False	False	False	False	False	False	False
Top $\mathcal{K}$	10	10	10	10	10	10	10	10	10
Found	3103	1520	3782	3186	1029	1094	3186	1520	3144
Length	3800	3800	3800	3800	3800	3800	3800	3800	3800
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.65	0.5044	0.9629	0.6637	0.4966	0.5004	0.6637	0.5356	0.845
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.5755	0.5131	0.5397	0.5826	0.5145	0.5181	0.5826	0.5395	0.5529
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.4949	0.4988	0.499	0.5031	0.5011	0.4958	0.5031	0.4875	0.4968
Pos	30399	77816	5	26607	96394	72004	26607	92949	26812
$P@K$	0.8165	0.4	0.9952	0.8384	0.2707	0.2878	0.8384	0.4	0.8273
$MRR@K$	0.8165	0.4	0.9952	0.83842	0.2707	0.2752	0.8384	0.4	0.8273
$R@\mathcal{K}^n$	0.0878	0.7289	0.5176	0.1821	0.1405	0.2147	0.0623	0.0244	0.765
$P@\mathcal{K}^n$	0.0008	0.0072	0.0051	0.0006	0.0013	0.0021	0.0006	0.0002	0.0075
$F_1@\mathcal{K}^n$	0.0017	0.0142	0.0101	0.0012	0.0027	0.0042	0.0012	0.0004	0.015

Table A.7.: Results and setup of experiment B for the data set *Diginetica* with $\mathcal{K} = 10$. STR stands for STRUCTURAL, SES stands for SESSION and HIS stands for HISTORY.

Hyperparameter	10
Experiment	C
MIN-COUNT-USER	5
MIN-COUNT-ITEM	5
REMOVE-COMMON-ITEMS	0.05
REMOVE-COMMON-USERS	0.05
RUN-MODE	SESSION
NEGATIVE-SAMPLER	MOST
USE-CATEGORY-SWITCH	0
USE-SESSION-TIME-SAVE	1
Use MinMax	False
Top $\mathcal{K}$	10
Found	1651
Length	3800
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.4848
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.5445
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.4934
Pos	95885
$P@K$	0.4344
$MRR@K$	0.4344
$R@\mathcal{K}^n$	0.4213
$P@\mathcal{K}^n$	0.0042
$F_1@\mathcal{K}^n$	0.0084

Table A.8.: Results and setup of experiment C for the data set *Diginetica* with $\mathcal{K} = 10$

Hyperparameter	11	12	13	14
Experiment	D	D	D	D
MIN-COUNT-USER	5	5	5	5
MIN-COUNT-ITEM	5	5	5	5
REMOVE-COMMON-ITEMS	0.05	0.05	0.05	0.05
REMOVE-COMMON-USERS	0.05	0.05	0.05	0.05
RUN-MODE	HISTORY	HISTORY	HISTORY	HISTORY
NEGATIVE-SAMPLER	MOST	MOST	MOST	LESS
USE-CATEGORY-SWITCH	0	0	1	0
USE-SESSION-TIME-SAVE	0	0	0	0
Use MinMax	False	True	False	False
Top $\mathcal{K}$	10	10	10	10
Found	3070	3685	3578	2106
Length	3800	3800	3800	3800
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.6281	0.6775	0.915	0.5742
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.5525	0.5316	0.5473	0.5614
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.4813	0.4805	0.5022	0.5063
Pos	21729	2388	9504	55710
$P@K$	0.8078	0.9697	0.9415	0.5542
$MRR@K$	0.8078	0.9689	0.9415	0.5536
$R@\mathcal{K}^n$	0.0971	0.1276	0.9489	0.1863
$P@\mathcal{K}^n$	0.001	0.0013	0.0093	0.0018
$F_1@\mathcal{K}^n$	0.0019	0.0025	0.0186	0.0037

Table A.9.: Results and setup of experiment D for the data set *Diginetica* with $\mathcal{K} = 10$

Hyperparameter	15	16	17	18
Experiment	E	E	E	E
MIN-COUNT-USER	5	3	5	3
MIN-COUNT-ITEM	5	3	5	3
REMOVE-COMMON-ITEMS	0.1	0.05	0.1	0.05
REMOVE-COMMON-USERS	0.1	0.05	0.1	0.05
RUN-MODE	SESSION	SESSION	HISTORY	HISTORY
NEGATIVE-SAMPLER	MOST	MOST	MOST	MOST
USE-CATEGORY-SWITCH	0	0	1	1
USE-SESSION-TIME-SAVE	1	1	0	0
Use MinMax	False	False	False	False
Top $\mathcal{K}$	10	10	10	10
Found	200	2060	199	2060
Length	200	15000	200	6639
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.8249	0.4428	0.9493	0.5489
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.5417	0.6001	0.5365	0.5745
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.5088	0.4981	0.4978	0.5037
Pos	1	101640	1	74300
$P@K$	1.0	0.13733	0.995	0.4426
$MRR@K$	1.0	0.13733	0.995	0.4426
$R@\mathcal{K}^n$	0.77	0.2017	0.885	0.1075
$P@\mathcal{K}^n$	0.0076	0.002	0.0087	0.001
$F_1@\mathcal{K}^n$	0.0151	0.004	0.01735	0.0021

Table A.10.: Results and setup of experiment E for the data set *Diginetica* with $\mathcal{K} = 10$

A.3. Retailrocket

In Table A.11 the results and the experimental setup of experiment A are shown for the data set *RetailRocket* with $\mathcal{K} = 10$. Similarly, Table A.12 represents experiment B, Table A.13 represents experiment C, Table A.14 represents experiment D, and Table A.11 represents experiment E. In these tables, n_{hit} indicates whether the score of the target item t is within the first $\mathcal{K}$, which can be described by $y_t > y_{\mathcal{K}}$. Where y_t is the target item score, $\frac{y_t}{|X_{\text{test}}|}$ is the mean of all scores within the test data set. Furthermore, $\frac{y_{\mathcal{K}}}{|X_{\text{test}}|}$ represents the average of the highest values of $\mathcal{K}$ and $\frac{y}{|X_{\text{test}}|}$ describes the average in general. In addition, $\frac{\text{pos}}{|X_{\text{test}}|}$ reflects the average position within y. Also, LESS stands for LESS-SIMILAR and MOST stands for MOST-SIMILAR. The other points are as defined by subsection 2.5.2.

Hyperparameter	1	2	3
Experiment	A/B	A/B	A/B
MIN-COUNT-USER	2	2	2
MIN-COUNT-ITEM	2	2	2
REMOVE-COMMON-ITEMS	0.05	0.05	0.05
REMOVE-COMMON-USERS	0.05	0.05	0.05
RUN-MODE	STRUCTURAL	SESSION	HISTORY
NEGATIVE-SAMPLER	MOST	MOST	MOST
USE-CATEGORY-SWITCH	1	1	1
USE-SESSION-TIME-SAVE	1	1	1
Use MinMax	False	False	False
Top $\mathcal{K}$	10	10	10
Found	45942	50400	49680
Length	50400	50400	50400
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.8865	1.0	0.8854
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.6782	0.5581	0.6368
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.5235	0.4971	0.4999
Pos	30890	1	3475
$P@K$	0.9115	1.0	0.9857
$MRR@K$	0.9115	1.0	0.9857
$R@\mathcal{K}^n$	0.0505	1.0	0.1082
$P@\mathcal{K}^n$	0.0005	0.0099	0.001
$F_1@\mathcal{K}^n$	0.0009	0.0196	0.0021

Table A.11.: Results and setup of experiment A for the data set *RetailRocket* with $\mathcal{K} = 10$

Hyperparameter	1	2	3	4	5	6	7	8	9
Experiment	A/B	A/B	A/B	B	B	B	B	B	B
MIN-COUNT-USER	2	2	2	2	2	2	2	2	2
MIN-COUNT-ITEM	2	2	2	2	2	2	2	2	2
REMOVE-COMMON-ITEMS	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
REMOVE-COMMON-USERS	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
RUN-MODE	STR	SES	HIS	STR	SES	HIS	STR	SES	HIS
NEGATIVE-SAMPLER	MOST	MOST	MOST	RANDOM	RANDOM	RANDOM	LESS	LESS	LESS
USE-CATEGORY-SWITCH	1	1	1	1	1	1	1	1	1
USE-SESSION-TIME-SAVE	1	1	1	1	1	1	1	1	1
Use MinMax	False	False	False	False	False	False	False	False	False
Top $\mathcal{K}$	10	10	10	10	10	10	10	10	10
Found	45942	50400	49680	9951	50395	8725	47168	50400	44378
Length	50400	50400	50400	50400	50400	50400	50400	50400	50400
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.8865	1.0	0.8854	0.5306	0.9999	0.5325	0.8848	1.0	0.8718
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.6782	0.5581	0.6368	0.6573	0.5855	0.6507	0.6618	0.5397	0.6441
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.5235	0.4971	0.4999	0.4822	0.4968	0.4772	0.5171	0.5034	0.4975
Pos	30890	1	3475	177482	2	168952	21903	1	40376
$P@K$	0.9115	1.0	0.9857	0.1974	0.9999	0.1731	0.9358	1.0	0.8805
$MRR@K$	0.9115	1.0	0.9857	0.1932	0.9998	0.1719	0.9358	1.0	0.8804
$R@\mathcal{K}^n$	0.0505	1.0	0.1082	0.5221	0.9963	0.6118	0.0575	1.0	0.0506
$P@\mathcal{K}^n$	0.0005	0.0099	0.001	0.0051	0.0098	0.006	0.0005	0.0099	0.0005
$F_1@\mathcal{K}^n$	0.0009	0.0196	0.0021	0.0102	0.0195	0.0119	0.0011	0.0196	0.0009

Table A.12.: Results and setup of experiment B for the data set *RetailRocket* with $\mathcal{K} = 10$. STR stands for STRUCTURAL, SES stands for SESSION and HIS stands for HISTORY.

Hyperparameter	10
Experiment	C
MIN-COUNT-USER	2
MIN-COUNT-ITEM	2
REMOVE-COMMON-ITEMS	0.05
REMOVE-COMMON-USERS	0.05
RUN-MODE	SESSION
NEGATIVE-SAMPLER	MOST
USE-CATEGORY-SWITCH	0
USE-SESSION-TIME-SAVE	1
Use MinMax	False
Top $\mathcal{K}$	10
Found	50400
Length	50400
$\frac{y}{\lvert X_{\text{test}}\rvert}$	1.0
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.5545
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.4999
Pos	1
$P@K$	1.0
$MRR@K$	1.0
$R@\mathcal{K}^n$	1.0
$P@\mathcal{K}^n$	0.0099
$F_1@\mathcal{K}^n$	0.0196

Table A.13.: Results and setup of experiment C for the data set *RetailRocket* with $\mathcal{K} = 10$

Hyperparameter	11	12	13	14
Experiment	D	D	D	D
MIN-COUNT-USER	2	2	2	2
MIN-COUNT-ITEM	2	2	2	2
REMOVE-COMMON-ITEMS	0.05	0.05	0.05	0.05
REMOVE-COMMON-USERS	0.05	0.05	0.05	0.05
RUN-MODE	HISTORY	HISTORY	HISTORY	HISTORY
NEGATIVE-SAMPLER	MOST	MOST	MOST	LESS
USE-CATEGORY-SWITCH	0	0	1	0
USE-SESSION-TIME-SAVE	0	0	0	0
Use MinMax	False	True	False	False
Top $\mathcal{K}$	10	10	10	10
Found	50400	50148	50400	50400
Length	50400	50400	50400	50400
$\frac{y}{\lvert X_{\text{test}}\rvert}$	1.0	0.8816	1.0	1.0
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.5396	0.0431	0.5659	0.5426
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.4979	0.3857	0.5015	0.5079
Pos	1	71	1	1
$P@K$	1.0	0.995	1.0	1.0
$MRR@K$	1.0	0.995	1.0	1.0
$R@\mathcal{K}^n$	1.0	0.0431	1.0	1.0
$P@\mathcal{K}^n$	0.0099	0.0004	0.0099	0.0099
$F_1@\mathcal{K}^n$	0.0196	0.0008	0.0196	0.0099

Table A.14.: Results and setup of experiment D for the data set *RetailRocket* with $\mathcal{K} = 10$

Hyperparameter	15	16	17	18
Experiment	E	E	E	E
MIN-COUNT-USER	2	3	2	3
MIN-COUNT-ITEM	2	3	2	3
REMOVE-COMMON-ITEMS	0.1	0.05	0.1	0.05
REMOVE-COMMON-USERS	0.1	0.05	0.1	0.05
RUN-MODE	SESSION	SESSION	HISTORY	HISTORY
NEGATIVE-SAMPLER	MOST	MOST	MOST	MOST
USE-CATEGORY-SWITCH	0	0	0	0
USE-SESSION-TIME-SAVE	1	1	0	1
Use MinMax	False	False	False	False
Top $\mathcal{K}$	10	10	10	10
Found	15000	60400	15000	60400
Length	15000	60400	15000	60400
$\frac{y}{\lvert X_{\text{test}}\rvert}$	1.0	1.0	1.0	1.0
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.5478	0.5708	0.5444	0.6218
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.5005	0.4946	0.5017	0.501
Pos	1	1	1	1
$P@K$	1.0	1.0	1.0	1.0
$MRR@K$	1.0	1.0	1.0	1.0
$R@\mathcal{K}^n$	1.0	1.0	1.0	1.0
$P@\mathcal{K}^n$	0.0099	0.0099	0.0099	0.0099
$F_1@\mathcal{K}^n$	0.0196	0.0196	0.0196	0.0196

Table A.15.: Results and setup of experiment E for the data set *RetailRocket* with $\mathcal{K} = 10$

A.4. YooChoose

In Table A.16 the results and the experimental setup of experiment A are shown for the data set *YooChoose* with $\mathcal{K} = 10$. Similarly, Table A.17 represents experiment C and Table A.18 represents experiment E. In these tables, n_{hit} indicates whether the score of the target item t is within the first $\mathcal{K}$, which can be described by $y_t > y_{\mathcal{K}}$. Where y_t is the target item score, $\frac{y_t}{|X_{\text{test}}|}$ is the mean of all scores within the test data set. Furthermore, $\frac{y_{\mathcal{K}}}{|X_{\text{test}}|}$ represents the average of the highest values of $\mathcal{K}$ and $\frac{y}{|X_{\text{test}}|}$ describes the average in general. In addition, $\frac{\text{pos}}{|X_{\text{test}}|}$ reflects the average position within y. The other points are as defined by subsection 2.5.2.

Hyperparamter	1	2	3
Experiment	A	A	A
MIN-COUNT-USER	5	5	5
MIN-COUNT-ITEM	5	5	5
REMOVE-COMMON-ITEMS	0.05	0.05	0.05
REMOVE-COMMON-USERS	0.05	0.05	0.05
RUN-MODE	STRUCTURAL	SESSION	HISTORY
NEGATIVE-SAMPLER	RANDOM	RANDOM	RANDOM
USE-CATEGORY-SWITCH	1	1	1
USE-SESSION-TIME-SAVE	1	1	1
Use MinMax	False	False	False
Top $\mathcal{K}$	10	10	10
Found	76574	76587	76587
Length	76600	76600	76600
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.9996	0.9998	0.9998
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.6965	0.7005	0.7005
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.5052	0.5164	0.5164
Pos	6.6385	3.9487	3.9487
$P@K$	0.9996	0.9998	0.9998
$MRR@K$	0.9996	0.9996	0.9996
$R@\mathcal{K}^n$	1	1	1
$P@\mathcal{K}^n$	0.0099	0.0099	0.0099
$F_1@\mathcal{K}^n$	0.0196	0.0166	0.0196

Table A.16.: Results and setup of experiment A for the data set *YooChoose* with $\mathcal{K} = 10$

Hyperparamter	10
Experiment	C
MIN-COUNT-USER	5
MIN-COUNT-ITEM	5
REMOVE-COMMON-ITEMS	0.05
REMOVE-COMMON-USERS	0.05
RUN-MODE	SESSION
NEGATIVE-SAMPLER	RANDOM
USE-CATEGORY-SWITCH	0
USE-SESSION-TIME-SAVE	1
Use MinMax	False
Top $\mathcal{K}$	10
Found	76587
Length	76600
$\frac{y}{\lvert X_{\text{test}}\rvert}$	0.9999
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.7487
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.5144
Pos	1.2118
$P@K$	0.9999
$MRR@K$	0.9999
$R@\mathcal{K}^n$	1
$P@\mathcal{K}^n$	0.0099
$F_1@\mathcal{K}^n$	0.0196

Table A.17.: Results and setup of experiment C for the data set *YooChoose* with $\mathcal{K} = 10$

Hyperparamter	15	16
Experiment	E	E
MIN-COUNT-USER	5	3
MIN-COUNT-ITEM	5	3
REMOVE-COMMON-ITEMS	0.1	0.05
REMOVE-COMMON-USERS	0.1	0.05
RUN-MODE	SESSION	SESSION
NEGATIVE-SAMPLER	RANDOM	RANDOM
USE-CATEGORY-SWITCH	0	0
USE-SESSION-TIME-SAVE	1	1
Use MinMax	False	False
Top $\mathcal{K}$	10	10
Found	6600	178200
Length	6600	178200
$\frac{y}{\lvert X_{\text{test}}\rvert}$	1.0	1.0
$\frac{y_{\mathcal{K}}}{\lvert X_{\text{test}}\rvert}$	0.5742	0.7194
$\frac{y_t}{\lvert X_{\text{test}}\rvert}$	0.5087	0.4772
Pos	1.0	1.0
$P@K$	1.0	1.0
$MRR@K$	1.0	1.0
$R@\mathcal{K}^n$	1	1
$P@\mathcal{K}^n$	0.0099	0.0099
$F_1@\mathcal{K}^n$	0.0196	0.0196

Table A.18.: Results and setup of experiment E for the data set *YooChoose* with $\mathcal{K} = 10$