POLITECNICO DI MILANO
DEIB
DOCTORAL PROGRAM IN INFORMATION TECHNOLOGY

ALGORITHMS FOR SEQUENCE-AWARE RECOMMENDER SYSTEMS

Doctoral Dissertation of:
Massimo Quadrana

Supervisor:
Prof. Paolo Cremonesi

Tutor:
Prof. Stefano Ceri

The Chair of the Doctoral Program:
Prof. Andrea Bonarini

2017 – XXIX

Sommario

I sistemi di raccomandazione sono sicuramente tra le applicazioni di maggiore successo del data mining e machine learning; molte innovazioni tecnologiche significative su questo fronte sono state sviluppate negli ultimi due decenni. La ricerca accademica in questo campo è stata fortemente sospinta dalla disponibilità di grandi dataset composti da matrici user-item. La vasta maggioranza di questi lavori si è quindi focalizzata su di un'astrazione del problema basata su singole interazioni user-item. Il problema della raccomandazione si presenta quindi come completamento di matrici fortemente sparse, in cui le interazioni user-item mancanti devono essere predette.

Ciò nonostante, in molti domini si registrano multiple interazioni di tipo diverso tra user e items nel corso del tempo. La maggior parte degli algoritmi ottimizzati per questa formulazione del problema non sono in grado di utilizzare l'informazione contenuta nelle *sequenze ordinate di interazioni* che sono frequentemente registrate nei log di molte applicazioni reali. Esistono inoltre domini nei quali i prodotti devono essere raccomandati in un certo ordine. Anche queste situazioni non sono gestite dagli algoritmi basati sulle sole matrici user-item.

Per rispondere a queste esigenze, è stata recentemente introdotta una nuova classe di algoritmi detti *sequence-aware recommender systems* (SARS). Questi algoritmi possono gestire l'informazione contenuta nei log di interazioni degli utenti senza dover ricorrere ad ulteriori astrazioni come quella della matrice user-item.

Questa tesi si focalizza sullo studio e definizione di nuovi algoritmi di

raccomandazione sequence-aware e sulle rispettive applicazioni. Viene inizialmente presentata una caratterizzazione dettagliata del problema, delle sue relazioni e differenze rispetto ad altri problemi di raccomandazione correlati (nello specifico, la raccomandazione basata sulla matrice user-item, i sistemi di raccomandazione context-aware e time-aware). Viene infine fornita un'analisi dello stato dell'arte, degli algoritmi esistenti e delle procedure di valutazione.

La seconda parte si focalizza su due problemi specifici, quelli di raccomandazione session-based e session-aware. Questi problemi hanno ricevuto particolare attenzione da parte della comunità solo di recente data la loro rilevanza in molti scenari pratici. Viene inizialmente presentato uno user-study atto a validare l'utilità di algoritmi sequence-aware personalizzati nel contesto delle prenotazioni di hotel. Dopodiché vengono presentati due nuovi algoritmi per la raccomandazione session-based e session-aware. In questi scenari è disponibile la sequenza di azioni più recenti dell'utente (quelli relativi alla sessione corrente); l'obiettivo è quello di determinare gli item rilevanti per l'utente nella sessione corrente, considerando anche gli interessi storici dello stesso quando questi sono disponibili. A tale scopo abbiamo studiato modelli basati su Recurrent Neural Network (RNN), modelli neurali studiati espressamente per processare sequenze di informazioni. I nostri esperimenti mostrano che nuovi sistemi di raccomandazione sequence-aware basati su RNN sono efficaci in numerosi scenari applicativi reali, quali la generazione di raccomandazioni session-based basate su descrittori dei prodotti, la personalizzazione delle raccomandazioni session-based per utenti che riutilizzano il servizio, la raccomandazione di stazioni musicali e la generazione automatica di playlist. Questi modelli ci hanno permesso di studiare anche l'importanza dell'ordine delle trace in una playlist, un problema ancora largamente irrisolto per la comunità del Music Information Retrieval.

Gli approcci presentati in questa tesi sono stati validati utilizzando diversi grandi dataset di domini differenti, quali video, annunci pubblicitari e lavorativi, hotel e musica. Viene inoltre presentato un nuovo dataset per la raccomandazione musicale all'interno delle sessioni di ascolto degli utenti. I risultati sperimentali mostrano la validità dei modelli sequence-aware presentati in questa tesi.

Abstract

RECOMMENDER SYSTEMS are one of the most successful applications of data mining and machine learning technology in practice and significant technological advances have been made over the last two decades. Academic research in the field in the recent past was strongly fueled by the increasing availability of large datasets containing user-item rating matrices. Many of these works were therefore based on a problem abstraction where only one single user-item interaction is considered in the recommendation process. The recommendation problem is therefore framed as matrix-completion, in which the missing entries in the user-interaction matrix have to be predicted.

In many application domains, however, multiple user-item interactions of different types can be recorded over time. Most algorithms that are optimized for this particular problem setting cannot make use of the rich information that is hidden in the sequentially-ordered *user interaction logs* which are often available in practical applications. In addition, there are application domains, in which the items have to be recommended in a certain order. Such situations are typically not covered as well in research setups that rely on a user-item rating matrix.

To address this problem, in the recent years researchers have developed a new breed of algorithms named *sequence-aware recommender systems* (SARS). Such algorithms can handle the information in user interaction logs by design without resorting on abstractions such as the user-item matrix.

This thesis focuses on the study of novel algorithms for sequence-aware

recommender systems and their applications. We first provide a characterization of the problem; we highlight the relations and differences with respect to other related recommendation problems, namely recommendation based on matrix-completion, and with respect to context-aware and time-aware recommender systems. We provide an in-depth review of the state of the art, a categorization of the existing approaches and evaluation methodologies. We then focus on the problems of session-based and session-aware recommendation. These problems have gained attention recently, given their proximity with many real-world recommendation scenarios. We first validate the usefulness of personalized sequence-aware recommendations in session-based scenarios through a user study run in the hotel booking domain. We then present novel sequence-aware algorithms for session-based and session-aware recommendation. In such a setting, we are given the sequence of the most recent actions of a user and the problem is to find items that are relevant in the context of the session and, when historical information on the user is available, that also match the user's general interests and taste. In particular, we investigate models based on Recurrent Neural Networks (RNN), the neural network configuration of choice for processing sequentially-ordered data. We show the effectiveness of sequence-aware recommenders based on RNNs in several real-life scenarios, namely session-based recommendation with rich product descriptors, personalized session-based recommendation for returning users, music station recommendation and automated playlist generation. We also investigate the importance of the track order in automated playlist generation, shedding some light on this long debated issue by the Music Information Retrieval community.

In our experimental evaluation, we empirically evaluate the proposed models on large datasets from several domains, namely video, classified advertisement, hotel, job and music recommendation. We also present a novel large-scale dataset for music recommendation over user listening sessions. The empirical results show that our sequence-aware models are indeed effective in several session-based recommendation scenarios in terms of recommendation accuracy.

Contents

Part I

Background

CHAPTER 1

Introduction

1.1 Motivations

Recommender Systems (RS) are software applications that support users in finding items of interest within larger collections of objects, often in a personalized way. Today, such systems are used in a variety of application domains, including for example e-commerce or media streaming, and receiving automated recommendations of different forms has become a part of our daily online user experience. Internally, such systems analyze the past behavior of individual users or of a user community as a whole to detect patterns in the data. On typical online sites, various types of relevant actions of a user can be recorded, e.g., that a user views an item or makes a purchase, and several of the actions of a single user may relate to the same item. These recorded actions and the detected patterns are then used to compute recommendations that match the preference profiles of individual users.

Along with the success of RS in practice, academic research in the field has made tremendous progress in the past decade in terms of the development of new techniques to accurately predict the relevance of items for individual users. In terms of the data that is used in academic settings to design

and evaluate recommendation algorithms, the most common setup is to use (publicly available) datasets that contain a user-item *rating matrix*. Such a matrix contains at most one explicitly-stated or implicitly-derived preference statement for each user-item combination, and the computational task of a recommender often is to compute the missing values in the matrix.

While such a research setup helps to abstract of the particularities of a domain and supports the reproducibility of research results, it potentially also oversimplifies the underlying problem. Furthermore, algorithms that are optimized for this particular problem setting cannot make use of the rich information that is hidden in the sequentially-ordered user interaction logs which are often available in practical applications. In addition, there are application domains in which the items have to be recommended in a certain order. Such situations are typically not covered as well in research setups that rely on a user-item rating matrix.

A typical example problem setting in which sequentially-ordered user interaction logs are crucial for obtaining highly accurate recommendations is that of *session-aware recommendation*. In such a setting, we are given the sequence of the most recent actions of a user and the problem is to find items that not only match the user's general taste but are also relevant in the context of the session. The intent of the user is, therefore, strongly bounded to the current session, and recommendations must be dynamically adapted as the user's intent get sequentially revealed through her interaction with the system.

In some more extreme but very common scenarios, no information about the user's general taste is available to the system (*session-based recommendation*). This typically happens in contexts in which users rarely register and log-in to the system, like in many video streaming platforms for example, or in context with very low user return rates. In both cases, we must resort only on the handful of interactions of the user in the current session to generate recommendations. It becomes, therefore, crucially important to infer any possible detail about the user's interests and goals from the sequence of actions in the current session. To this end, relevant information can be gathered also from the characteristics of the items the user interacts with. The problem of context adaptation with sequence-aware recommenders in session-based and session-aware scenarios is the main subject of this thesis.

In some application domains even the *order of the elements* in the current session is relevant. In playlist recommendation, for example, there exists a long debated issue on whether the order of the tracks in the playlist, or at least their relative position, has any importance for the recommendation,

i.e. for predicting which tracks will be successively added to the playlist. In this thesis we consider the problem of automated playlist generation, in which tracks are sequentially added on the basis of the existing tracks in the playlist, and used sequence-aware models to shed some light on this question. We finally present a new large-scale dataset of music listening sessions, with the goal of helping the research in this field beyond what is achievable with the currently available data sources.

1.2 Goals

In this thesis we aim to develop novel algorithms for sequence-aware recommender systems. In particular, we focus our attention on a prominent problem in real-life scenarios, that of session-based recommendation. For such purpose, we worked to create novel and advanced machine learning algorithms that can naturally deal with large quantities of sequentially-ordered data from the logs of user activity, and generated effective recommendations in real-life scenarios. To this end, we have addressed the following research goals.

RG1: review the state of the art on sequence-aware recommender systems, in order to characterize the tasks and the existing algorithmic approaches to this problem. As stated in the motivation, most of the research in Recommender Systems considers the matrix-completion paradigm. However, a huge body of research has been recently done to extend or substitute this paradigm to address the need for generating recommendations from the logs of user actions. We aim at an extensive and rigorous survey of the state of the art in sequence-aware recommender systems to characterize the tasks, algorithms and evaluation procedures adopted in the literature so far.

RG2: measure the perceived impact of personalized recommendations in session-based recommendation scenarios through a user study. From the analysis in RG1, we noticed that the majority of the existing works employ sequence-aware recommenders for context-adaptation based on the last-N actions of the user. Until the most recent times, the problems of session-based and session-aware recommendation have been mostly neglected by the community. These two problems, however, are better representative of the way users interact with many online systems, and are therefore worth to be investigated in detail. We also noticed an overall lack of user studies in sequence-aware recommendation, excluded few exceptions [53, 96], and in session-based recommendation in general.

As a first step, we aimed at evaluating the perceived quality of person-

alized recommendations in a true session-based scenario through a user study in the hotel booking domain. We used a simple sequence-aware approach based on implicit elicitation combined with traditional collaborative and content-based RS. Once we assessed the effectiveness of the recommendations in session-based scenarios, we pursued the development of advanced sequence-aware algorithms for session-based and session-aware recommendation.

RG3: develop novel sequence-aware algorithms for real-world session-based recommendation scenarios, that can handle both new and returning users, and exploit item features as well. From our survey of the state of the art in RG1, we noticed the complete absence of sequence-aware solutions that consider item features as part of the modeling of the user activity within the session. Item features can provide precious details on the actual interests of the user in the current session, and can be extremely useful in session-based scenarios where no historical information on the user is available. Beside this, we noticed an almost complete absence of sequence-aware algorithms for session-aware recommendation. These algorithms are aimed at modeling the historical interests of the user together with her short-term goals within sessions, with the goal of providing personalized session-based recommendation suited to the long-term and short-term interest of the user [1].

We therefore worked on the development of new sequence-aware algorithms for session-based and session-aware recommendation. Supported by the recent breakthroughs in deep-learning models, we focused on models based on Recurrent Neural Networks. We target two orthogonal problems: (i) session-based recommendation enriched with item features, that can leverage the sequential dependencies between items and their features in modeling the user's activity in the current session; (ii) session-aware recommendation, that merges the long-term interests of the user with her activity within the session into a single, comprehensive, sequence-aware recommender system. We validate the performance of the proposed approaches over several large-scale datasets coming from real-world industrial application scenarios, using standard evaluation methodologies.

RG4: model the evolution of musical tastes through radio-station recommendation, verify that order matters in automated playlist generation, and foster the research on sequence-aware session-based music

[1] Notice that "personalization" in this context refers to the capability of the recommender to provide different recommendations to users that have performed exactly the same sequence of interactions in the current session. In "pure" session-based recommendation, instead, recommendations are customized solely on the basis of the actions of the user in the current session. Hence, two users that perform the same sequence of actions look exactly the same to the eyes of the RS in the latter case.

recommendation by releasing a new, large-scale music listening dataset. Again from our review of the state of the art in RG1, we noticed that music is one of the application domains in which sequence-aware recommenders are widely employed, e.g. to generate automatically plalyists and radio stations. With to respect this, we first study sequence-aware recommenders based on Recurrent Neural Networks as models to represent the evolution of user musical tastes through radio station recommendation. We then delve into the dynamic of automated playlist generation, and addressed one important question remains open in the domain of music recommendation, namely that of the importance of the order of songs in playlists. We therefore designed an experiment aimed at answering this question. Finally, to address the need for large datasets for session-based music listening recommendation, we present a novel, open, large-scale session-based dataset for music listening recommendation.

1.3 Contributions

This thesis work has resulted in several contributions to the state of the art in sequence-aware recommender systems, which is summarized next.

In Chapter 2 we provide a comprehensive, in-depth review of the state of the art in sequence-aware recommender systems. We present a **characterization of the problem with respect to other areas of recommender systems**, we discuss about the **main tasks and subtasks** targeted by sequence-aware recommenders, and we provide a detailed **review of the algorithmic approaches** used in the literature so far. We categorize the existing works based on their task and algorithmic family. Finally, we review the **offline evaluation procedures** for sequence-aware recommender systems.

In Chapter 3 we present a **case-study on session-based recommendation for hotel bookings**. We investigate a methodology based on implicit elicitation to leverage traditional recommender algorithms, such as collaborative filtering and content-based algorithms, to address session-based recommendation in an hotel booking scenario with new users. We perform an online study and show the effectiveness of recommenders over trivial non-personalized and editorial baselines in a strongly popularity biased scenario.

In Chapter 4 we present a **novel sequence-aware recommender based on Recurrent Neural Networks (RNN) for feature rich session-based recommendation**. Our work differs from the existing works in the use of item features along with item identifiers to model the sequential aspects in the activity of the user in the current session, with final goal of improving

the accuracy of the recommendation. We show that relevant sequential features can be extracted also from the "raw" content of items (i.e. not using abstract item descriptors such as metadata). We prove that item features can be effectively leveraged to improve the ranking quality through extensive evaluation on two industrial datasets for session-based recommendation scenarios with new users.

In Chapter 5 we present a **novel sequence-aware recommender based on Hierarchical RNNs for personalized session-based recommendation for returning users**. The proposed solution is capable of modeling the evolution of user interests over time, and to use the long-term user's interests to personalize the recommendations at session level. Differently from the existing works that decouple the long-term and short-term models of the user, our method seamlessly models both aspects into a single, comprehensive framework. The experimental results on two industrial datasets show the effectiveness of personalization in session-based recommendation, both in scenarios in which the user interests tend to be very repetitive (i.e. having weak contextual effects) or when they are strongly bounded to the current session (i.e. having strong contextual effects).

In Chapter 6 we study the effectiveness of sequence-aware recommender in **modeling the evolution of user musical tastes**. We consider station recommendation (i.e., the recommendation of radio station, akin to whole, infinite playlists) as a proxy for the user musical interests, and compare several sequence-aware recommenders from the state-of-the-art in this task. Our experimental results show that RNN-based models outperform the other competitive approaches in terms of sequential recommendation quality when the user history – expressed as the number stations available in the user profile – is sufficiently long. Moreover, we show that RNNs suggest more tracks in the long-tail of musical tastes than other approaches. We exploit this fact into by combining the predictions from several classifier into a Learning-to-Rank framework. Our results show significant improvements in the ranking quality when the recommendations generated by the RNN model are added to the pool of candidate items.

In Chapter 7 we investigate the **role of song ordering in automated playlist generation**. This is a long debated question that has not received enough agreement by the Music Information Retrieval community. We designed an experiment based on randomized playlists and used sequence-aware models based on RNNs to address this question. Our results show that the order of songs has negligible impact on the recommendation quality, and that RNN-based models are robust to new, unseen (randomized) sequences of songs.

In Chapter 8 we present a **new large-scale dataset for session-based music and playlist recommendation**. With respect to existing datasets, our dataset has unprecedented size and it is already partitioned into user sessions.

1.4 List of Publications

Journal publications

1. Deldjoo Y., Elahi M., Cremonesi P., Garzotto F., Piazzolla P., **Quadrana M.** (2016). Content-based video recommendation system based on stylistic visual features. *Journal of Data Semantics*

2. **Quadrana M.**, Bifet A., Gavaldà R. (2015). An efficient closed frequent itemset miner for the MOA stream mining system. *AI Communications* 28.1

Conference publications

1. **Quadrana M.**, Karatzoglou A., Hidasi B., Cremonesi P. (2017). Personalizing Session-based Recommendation with Hierarchical Recurrent Neural Networks. *In Proceedings of the 11th ACM conference on Recommender systems (RecSys 2017).*

2. Cella L., Cereda S., **Quadrana M.**, Cremonesi P. (2017). Deriving Item Features Relevance from Past User Interactions. *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization (UMAP 2017).*

3. Hidasi B., **Quadrana M.**, Karatzoglou A., Tikk D. (2016). Parallel Recurrent Neural Network Architectures for Feature-rich Session-based Recommendations. *In Proceedings of the 10th ACM conference on Recommender systems (RecSys 2016).*

4. Deldjoo Y., Elahi M., **Quadrana M.**, Cremonesi P., Garzotto F. (2015). Toward Effective Movie Recommendations Based on Mise-en-Scène Film Styles. *In Proceedings of the 11th Biannual Conference on Italian SIGCHI Chapter (CHItaly 2015).*

5. Cremonesi P., **Quadrana M.**. (2014). Cross-domain Recommendations without Overlapping Data: Myth or Reality?. *In Proceedings of the 8th ACM Conference on Recommender systems (RecSys 2014).*

6. Cremonesi P., Garzotto F., **Quadrana M.**. (2013). Evaluating top-n recommendations "when the best are gone". *In Proceedings of the 7th ACM conference on Recommender systems (RecSys 2013)*.

Posters

1. Vall A., **Quadrana M.**, Schedl M., Widmer G., Cremonesi P. (2017). The Importance of Song Context in Music Playlists: Enabling Recommendations in the Long Tail. *ACM RecSys 2017*

2. Cremonesi P., Garzotto F., Pagano R., **Quadrana M.** 2014. Recommending without short head. *In Proceedings of the companion publication of the 23rd international conference on World wide web companion (WWW 2014)*.

3. Turrin R., **Quadrana M.**, Pagano R., Paolo C., Andrea Condorelli. 30Music listening and playlists dataset. *ACM RecSys 2015*

1.5 Structure

This thesis is structured as follows:

Part I: Background

- Chapter 1 introduces this thesis by presenting motivations, research goals, contributions and the publications related to this thesis.

- Chapter 2 presents the sequence-aware recommenders systems by first providing an intuitive definition of the problem; we then characterized sequence-aware recommenders in terms of their inputs, outputs, computational tasks and relations with other areas in recommender systems. We identify the main tasks and subtasks addressed by sequence-aware recommenders, and categorize the state of the art according to this new classification. We then present a comprehensive review of the existing algorithms for sequence-aware recommender systems; we identify classes, sub-classes and families of algorithms, and categorize the state of the art according to our taxonomy. Finally, we review the existing data partitioning techniques and evaluation protocols for sequence-aware recommendation that are used throughout this thesis.

Part II: User Study

- Chapter 3 presents an experimental study on session-based recommendation for hotel booking.

Part III: Novel Algorithms

- Chapter 4 presents a novel algorithm for session-based recommendation with item features based on Recurrent Neural Networks. The proposed approach enhances the existing session-based models based on RNNs by models item identifiers and feature vectors in parallel by means of novel alternating training procedures.
- Chapter 5 presents a novel algorithm for personalized session-based recommendation based on Hierarchical RNNs. The proposed solution models the evolution of the user interests across sessions and leverages such information at session level seamlessly, providing an effective way of personalizing recommendations for returning users.

Part IV: Sequence-Aware Recommendation in Music

- Chapter 6 explores the use of sequence-aware recommenders to model the evolution of musical tastes through time.
- Chapter 7 compares the performance of several sequence-aware recommenders for automated playlist generation and investigates on the importance of the order of songs that are currently contained in a playlist in predicting the forthcoming tracks.
- Chapter 8 presents a novel large-scale dataset for the modeling of user listening activity at session level.

Part V: Conclusions

- Chapter 9 offers the conclusions and future works.

CHAPTER 2

Sequence-aware recommender systems

Recommender Systems (RS) are software applications that support users in finding items of interest within larger collections of objects, often in a personalized way. Today, such systems are used in a variety of application domains, including for example e-commerce or media streaming, and receiving automated recommendations of different forms has become a part of our daily online user experience.

Internally, such systems analyze the past behavior of individual users or of a user community as a whole to detect patterns in the data. On typical online sites, various *types* of relevant actions of a user can be recorded, e.g., that a user views an item or makes a purchase, and several of the actions of a single user may relate to the same item. These recorded actions and the detected patterns are then used to compute recommendations that match the preference profiles of individual users.

Along with the success of RS in practice, academic research in the field has made tremendous progress in the past decade in terms of the development of new techniques to accurately predict the relevance of items for individual users. In terms of the data that is used in academic settings to design and evaluate recommendation algorithms, the most common setup is to use (publicly available) datasets that contain a user-item *rating matrix*. Such a

matrix contains at most one explicitly-stated or implicitly-derived preference statement for each user-item combination, and the computational task of a recommender often is to compute the missing values in the matrix.

While such a research setup helps to abstract of the particularities of a domain and supports the reproducibility of research results, it potentially also oversimplifies the underlying problem. Furthermore, algorithms that are optimized for this particular problem setting cannot make use of the rich information that is hidden in the sequentially-ordered user interaction logs which are often available in practical applications. In addition, there are application domains, in which the items have to be recommended in a certain order. Such situations are typically not covered as well in research setups that rely on a user-item rating matrix.

A typical example problem setting in which sequentially-ordered user interaction logs are crucial for obtaining highly accurate recommendations is that of *session-based recommendation*. In such a setting, we are given the sequence of the most recent actions of a user and the problem is to find items that not only match the user's general taste but are also relevant in the context of the session. In some application domains even the *order of the elements* in the current session is relevant. For "next-track music recommendation" problems, for example, the goal is to find a track that is not too different from the last played track, e.g., in terms of the tempo. Generally, the goal in session-based recommendations is to extract patterns from sequentially-ordered interaction logs in an offline process which can then be matched with the current session to determine suitable recommendations.

There are, however, also other interesting patterns which can be derived from such interaction logs but are not tied to a specific user session. Based on the ordering and the timestamps of the user actions, we can for example detect *interest drifts* of individual users over time or detect short-term *popularity trends* in the community that can be exploited by recommendation algorithms. Furthermore, for the problem of recommending consumables like ink for a printer, we could use interaction logs to learn the best point in time to *remind* users to replenish their stock.

Finally, as mentioned above, there are application domains of recommender systems, where the recommendation of one item only makes sense after some other event has happened. E-commerce sites like Amazon.com sometimes provide recommendations under the label "Often bought together". However, when a user inspects an SD-card on the shop, recommending the purchase of a digital camera might not make much sense. Such ordering constraints might be weak or relatively strict (as in the case of the recommendation of accessories for another product) and be implemented

as part of the logic of a sequence-aware recommender. Furthermore, in particular the weak or implicit ordering constraints might be mined from sequentially-ordered interaction log data and considered in the recommendation process.

2.1 Intuitive definition

Sequence-aware recommendation deals with scenarios in which the *sequence* of recent interactions of the user is relevant in inferring her current interests or tastes and, in turn, to provide high quality recommendations. We provide here an intuitive, high-level definition for Sequence Aware Recommender System.

Definition 2.1.1. *A Sequence Aware Recommender System is a recommender system that models the dynamics in user's interaction by extracting of sequential features from the logs of historical user activity, with the ultimate goal of adapting to users' short-term needs and providing high quality suggestions on the following item(s) the user should interact.*

This definition draws a line between "traditional" item recommendation and sequence-aware recommendation. In traditional item recommendation, the recommender system has to infer the general user's preference over the individual items in order to suggest her items that she may like to explore. Recommendations are usually provided in the form of a list of items ordered by relevance [113]. The relevance of each item is computed independently from the other items the user has recently interacted with. Moreover, the interdependence between user's interactions is not considered as well.

We also emphasizes the centrality of the adaptation to the short-term intent and needs of the user in sequence-aware recommendation. Traditional item recommender systems tend to ignore the underlying structure of the user feedback, and this can severely limit the capability of the RS of capturing the user's intent unless it can be explicitly represented by some contextual variables, like in CARS [8]. This limitation becomes more evident when the most recent interactions of the user have a high influence on the actions she will perform next. We will discuss in detail on the differences between traditional item recommenders based on matrix-completion and sequence-aware recommenders in the next section.

Let us use an example to describe the problem. In Figure 2.1, the user is planning a trip and she has already booked a flight, a room in a hotel and rented a car. Intuitively, the next activity has to be something like visiting a museum or having fun at some attraction park since the user intent

Figure 2.1: *Example of sequential recommendation scenario. The user has already booked a flight, booked a room in a hotel and rented a car. What action she will most likely take next?*

is clearly to move around. Good recommendations are hence places in driving-distance from her destination. In this example, the intent of the user can be determined implicitly from the sequence of her actions. However, a traditional item recommender system can only infer the user preference over individual items (hotel, car, museum, park, etc.) but it has no way of inferring neither the user's intent, unless explicitly specified by the user.

Another scenario in which traditional item recommendation falls short is that of repeated item consumption, in which users generally interact with items multiple types, like in the consumption of multimedia content like music or videos for example. In automated playlist generation, for example, the music recommender system that has to suggest the next song(s) to append to a user generated playlist. Users usually listen tracks several times in different moments. User tastes can be elicited by a traditional item recommender system from their aggregated historical activity [5, 18]. However, a playlist is composed by songs that must play well in sequence and obey to some musicological 'coherence' properties [67]. By blindly adding songs picked from the user's favorites we can easily generate simple playlists that individually satisfy the user's taste, but lead to a musically unappealing experience when played in sequence. Instead, the knowledge over user's tastes must be combined with the characteristics of the sequence of tracks that are in the playlist in order to create an engaging listening experience that is coherent with the user's tastes [18]. We will investigate more in depth on the effectiveness of sequence-aware recommendation in automated playlist generation in Chapter 7.

2.2 Characterizing Sequence-Aware Recommender Systems

Sequence-aware recommendation problems are different from the traditional matrix-completion setup in a number of ways.

2.2.1 Inputs, Outputs, and Computational Tasks

Inputs The main input to sequence-aware recommendation problems is an ordered and usually time-stamped list of past user actions. Each action can be associated with one user of the system or be an anonymous action. Each action can furthermore be associated with one of the recommendable items. Finally, each action can be of one of several pre-defined *types* and each action, user, and item may have a number of additional attributes. In addition, a sequence-aware is given the point of time (e.g., in terms of the last user action or timestamp) for which a recommendation is sought for. Overall, the inputs can be considered as a sort of enriched clickstream data.

In the traditional matrix completion setup, all ratings are attached to one of the known users. We do not require this to be the case for sequence-aware recommenders. Anonymous user actions are not uncommon, e.g., in the e-commerce domain, where users are often not logged in. Nonetheless, relevant information can be extracted from past anonymous sessions. We also do not require that each action is related to an item, since, for example, relevant information can be extracted from the users' search or navigation behavior as well. Finally, in most application scenarios, each action will have an assigned action type (e.g., item-view, item-purchase, add-to-cart, etc.). And, depending on the domain, additional information might be available that describes further details of an action (e.g., whether an item was discounted when the action took place), the users (e.g., demographics), or the items (e.g., metadata features).

Generally, such forms of input data are available in many practical applications, e.g., in the form of application or web server logs, and we do not assume to have larger quantities of explicit ratings are available in sequence-aware recommender systems.

Outputs The output of a sequence-aware recommender are one or more ordered lists of items, where each of the items can have additional annotations.

In this general form, the outputs are similar to those of a traditional "item-ranking" recommendation setup.[1] However, in sequence-aware recommenders, the returned lists may have to be interpreted in different ways, depending on the specific application scenario. In some scenarios, like session-based recommendation, the interpretation can be as usual, i.e., the items represent *alternatives* for the user to consider next. In other scenarios,

[1] Rating predictions or relevance prediction scores *can* be the basis for ranking the items, but we do not require the existence of such scores.

however, the interpretation can be that the user should consider *all* recommendations and do this in the provided order. A typical example is the recommendation of a series of learning courses which can only be attended one after the other. In yet another scenario, the system might return multiple lists where each contains a ranked list of elements of a certain type. This could be the case when *complements* to a main item are recommended, e.g., a list of hotels and a list of car rental services that are displayed after the user has booked a flight. We will describe the application scenarios in more detail below.

Computational Tasks In order to filter and rank the items in the resulting output, different types of computations are usually done by sequence-aware recommenders, and the particularities of these calculations again depend on the specific application scenario. Most commonly, one task that is not present in traditional matrix completion setups is the automated identification of sequence-related patterns in the recorded user actions. This can be *sequential* patterns, where the order of the actions is relevant, or they can be *co-occurrence* patterns, where it is only important that two actions happened together, e.g., within the same session. In some cases, also *distance* patterns can be relevant, e.g., when the problem is to compute a good point in time to remind the user of something through a recommendation. Note that the corresponding patterns do not have to be explicit, as often done, e.g., in sequential pattern mining [85], but can be implicitly encoded in complex machine learning models as well.

Besides the identification of such patterns that are subsequently used in the recommendation task, another computational task of a sequence-aware recommender can be to reason about *order constraints*. Such constraints can be either prescribed and given for an application domain as strict constraints (e.g., in terms of a given curriculum for the learning course recommendation problem), given as heuristics (e.g., in terms of track transition rules for next-track music recommendation), or be implicitly derived from the given input data as a sort of weak constraints.

Finally, the patterns (or more generally, the learned models) that were identified in the data and the constraints have to be related with the point in time for which the recommendation is sought for. In a session-based recommender, one might consider the last few user actions and then look for past sessions that were similar to the current one. On the other hand, when a recommender is used as a reminder for the repeated purchase of consumables, the distance (in time) to the last purchase action of the user to the present time might be relevant.

2.2.2 Relation to Other Areas

Implicit-Feedback Recommender Systems Our characterization of the sequence-aware recommendation problem mainly targets scenarios in which we observe the individual and collective behavior of a user community over time instead of asking for explicit item ratings. Such explicit item ratings *can* be taken into account in a sequence-aware recommender as one of several types of user actions. One problem with explicit ratings however is that the point in time when users provide a rating can be quite different from the point in time when they consumed or purchased an item (e.g., when a user initially rates a bunch of movies when registering for a movie recommendation service). The sequence and timestamp of the ratings might easily mislead sequence-aware recommenders.

In the literature, the difference between implicit and explicit feedback recommender systems only lies in the type of the available preference signals. Many research works on implicit feedback algorithms are however focused on the matrix completion problem as well and do not consider multiple interactions over time as we do in our problem characterization.

Context-Aware and Time-Aware Recommender Systems In some of the application scenarios that we discuss in the next section, sequence-aware recommender systems can be considered as a special form of context-aware recommender systems. In particular in session-based recommendation scenarios the users' short-term intents, which can be estimated from their very last actions, can represent a crucial form of context information that should be taken into account when recommending [66]. Similarly, the current point in time can be seen as a relevant contextual factor in sequence-aware recommenders, e.g., when considering the repeated purchase of consumables.

Time-aware recommender systems specifically consider the timestamps that are associated with past user interactions in their models and adapt the recommendations to a given temporal context, see [20] for an overview. Time-aware recommender systems share a number of commonalities with sequence-aware recommenders, e.g., in terms of how we can benchmark different approaches in offline settings, as will be discussed later. The focus of sequence-aware recommenders is however often less on the exact point of time of past user interactions, but on the sequential order of events. Furthermore, a number of proposals on time-aware recommenders mainly rely on the matrix completion problem setting and, e.g., factor in the timestamps of the single user-item interactions into the recommendation process [73]. While this can serve similar purposes as sequence-aware recommenders,

e.g., detecting an interest drift in user preferences, many time-aware recommendation algorithms do not account for multiple user-item interactions over time.

2.3 Tasks

The recommendation quality can largely benefit from the exploitation of the features in sequences of user interactions. For example, the sequence of interactions in current session can be used to infer the goal of the user and, consequently, to provide recommendations that are tailored to that goal. Sequence-aware recommendation hence can be used for *context-adaptation*. From an in-depth analysis of the literature, context-adaptation is by far the most relevant task that is accomplished with sequence-aware recommenders (see Table 2.1).

Nevertheless, sequence-aware recommenders can be used for a variety of other *subtasks*, such as to find similar items (or substitutes) and complements in e-commerce, to generate list continuations (like in automated playlist generation), to detect individual and community trends and to identify repeated user behavior patterns. Finally, some important considerations can be made on the *ordering constraints* that rule the consumption of items in certain domains.

2.3.1 Context adaptation

Context adaptation in recommendation systems has been subject of extensive research. Users behave differently and exhibit different preferences depending on the context in which they interact with the system [2]. Traditional examples of contextual variables are time of the day, location, weather, mood and the goal of the user. According to the classification of [41], time, location and weather are examples of *representational context*, since the context can be explicitly defined by the combination of some observable attributes. On the other hand, variables like the mood of the user or her goal cannot be defined by means of any observable attribute, but they must be inferred from recent user's interactions. This type of context is said *interactional*. Additionally, some contextual variables like the goal of the user are strongly bounded to the current user session. This means that the RS may be asked to infer the context from just a handful of interactions performed by the user in the current session.

Let us consider the following example. Given a user who is navigating within the apps in her smart-phone, once opened the 'contacts' app, it is very likely that she will open the 'caller', 'messages' or 'e-mail' apps

next. However, she will unlikely launch the 'camera' app, regardless of the fact that she is currently in a park in a very sunny day. The sequence of apps opened by the user inform the system about her interactional context, whereas the current location and weather constitute the representational context. Additionally, the next action of the user will not be likely influenced by the apps she used the day before, i.e., her goal is bounded to the current session, or at least to the most recent apps opened. The most recent activity of the user is a proxy of the actual user's goal (e.g. to communicate with one of her contacts), and can be used to suggest to the user the right app to open next [84].

Sequence-aware recommenders can be used to infer the short-term goal of the user from the sequence of her most recent activities. Furthermore, the whole user history can be partitioned into *sessions* in which the user interacts with the system divided by periods with no user interactions. This is typical of domains like e-commerce or web browsing in which user sessions can be identified *explicitly* (e.g. with log-in/log-out actions) or *implicitly* (e.g. by analyzing the distribution of user activity over time and by "manually" partitioning the user log into sessions).

Context-adaptation can be obtained by extracting frequent sequential patterns from the historical activity of users and then by mapping it to the most recent user actions [84, 95, 98, 114, 145, 151]. Alternatively, sequence learning techniques can be used to generate a latent representation of the user's intent directly from the sequence of interactions [48, 57, 61, 65, 123, 127, 133, 148].

Context-adaptation with sequence-aware recommenders can be classified in the following three categories:

- *Last-N interactions*. This is the simplest form of context-adaptation and it is based on the sequence of the most recent N actions of the user. This is usually adopted when it is not possible to partition the user activity into sessions, or it simply does not make sense to do that. For example, in next check-in recommendation user movements are bounded to the very last few visited places or to the last one [26].

- *Session-based recommendation*. In this case recommendation depends exclusively on the current session. Session-based recommendation is common in systems having *new* or *anonymous* users – i.e. users not registered or logged into the system. For these kind of users, the only information that is available is the sequence of interactions in the current session. No historical data is available. This prevents from using classical personalization techniques, such as Collaborative Filtering,

since they rely on historical feedback. In this scenario the RS can rely only on the contextual information that can be extracted from the current session to generate recommendations. Therefore, it is crucially important to make full use of the information available in the current session to infer the goal of the user and to recommend the proper following interactions. This is a prominent problem in many practical scenarios like e-commerce [66] and news recommendation [48]. We will present significant enhancements to session-based recommendation based on sequence-aware methods in Chapters 4 and 5.

- *Session-aware recommendation*. In this case we instead focus on returning users, i.e. the user is not new to the the system and the recommender system can determine her historical, long-term preferences from her past activity. However, there is still the possibility that the long-term interests may not be aligned with the goal of the user in the current session. When users can consume items several times (e.g., in music or video-on-demand.), for example, the knowledge on historical activity of the user provides precious details on her tastes. However, knowing long-term user interests may not enough to provide high quality recommendations in the current session because the user can interact with the same item in different contexts or goals in mind. In music recommendation, for example, we may know that the user has a strong historical preference for classical music; despite that, the user may be interested in listening some hard-rock music in the current session, and suggesting Beethoven right after some hard-rock tracks is arguably a good recommendation. In other words, recommendations must be tailored on the short-term interests and historical, long-term interest used as additional source of information.

  The key challenge of session-aware recommendation is combining the long-term tastes and interests with short-term goals. Similarly to session-based recommendation, the short-term goal of the user can be inferred from the sequence of interactions in the current session. Still, the SARS now has to balance the long-term preferences of the user with her short-term interests.

  This problem can be addressed by jointly learning two models, one for long-term user profiling and a second one for the inference of the short-term goal of the user. The joint learning procedure allows to effectively model these two aspects together by, for example, optimizing a single objective function, like done in [112, 143] and [126]. We will present a method based on Hierarchical RNN that jointly models

the evolution of the user taste over time and the activity of the user at session level Chapter 5.

2.3.2 Subtasks

Beside context-adaptation, as described in the previous section, a variety of subtasks can be tackled with sequence-aware recommenders. We present here an overview of what we believe are possible subtasks that can be accomplished with SARS. Beware that many of them have received little attention from the community so far, so we were not able to find references in the literature for some of them. Nevertheless, we believe it is worth introducing them here also to shed light on the possible future developments of the field.

- *Find similar items:* In e-commerce settings, this problem corresponds to the task of finding *substitute* items, i.e., alternatives for the currently inspected item or the set of items inspected in a session. In another application domain the problem consists of recommending items that have similar features as a given object, e.g., a similar user on a social network, a next video to watch or a next track to listen to on a media streaming platform. Often, the selection of items to recommend is largely dependent on the very last user action (e.g., currently viewed item); the set of similar items is however usually based on patterns in the community (e.g., item co-occurrence patterns in sessions) as well and not only based on item metadata features.

- *Find complements:* In contrast to the last problem, the goal here is to find items that complement a given item. In the e-commerce scenario this for example would mean the recommendation of accessories for a given item. Note that in reality, recommendation lists often contain a mix of complements and substitutes and the optimal choice can depend on the domain [38]. Having complement items in a list can in general help users to discover new things.

- *Find a list continuation:* In this particular problem setting, the problem is to create an ordered list of recommendations which are all to be considered (consumed) by the user in the given order. In contrast to the *find similar items* task, the ordering of the recommended items is relevant. Sometimes the ordering of items of the current session is considered as well. A typical application scenario is the creation of automated playlist continuations of videos or musical tracks [67].

- *Find logically-next item(s):* In this case the next item to recommend is a application-specific "logical next step", given the last interaction(s) of the user. Examples of applications are next-query recommendation approaches [57,127] or next-app recommenders [7]. Usually, the logic of determining the next item follows more strict rules than, e.g., in heuristic-based next-track music recommendation scenarios.
- *Trend Detection:* Besides short-term adaptation, the detection of trends is another potential, but less explored, goal that can be accomplished by sequence-aware recommenders. We can distinguish between the following types of information that can be extracted from sequential log information to be used in the recommendation process.
  - *Community trends.* Considering the popularity of items within a user community can be important for successful recommendations in practice, e.g., in streaming media recommendation [51]. Since the popularity of items can change over time in different domains, sequence-aware recommenders can aim to detect and utilize popularity patterns in the interaction logs to improve the recommendations. Such trends can be long-term (e.g., things becoming outdated or out-of-fashion over time), seasonal, or reflect short-term and one-time popularity peaks. In the fashion domain, for example, considering the community trends of the last few days can represent a successful strategy when selecting items for recommendation [68].
  - *Individual trends.* Changes in the interest in certain items can also happen at an individual level. These interest changes can be caused when there is a "natural" interest drift, e.g., when users grow up, or when their preferences change over time, e.g., due to the influence of other people, due to exceptional events, or when they discover something new. An example of application is the modeling of the dynamics of the musical taste of users [97]. Changes of the individual behavior can also be caused by trends in the community, in particular when users are early adopters of new things. Note that session-aware recommendation, as discussed above, in some sense also implicitly considers short-term changes in the user behavior, but the focus is not explicitly on detecting and considering behavioral changes.
  - *Obsolete items.* As a special form of considering trends in the individual and collective user behaviour, sequence-aware recommenders can aim at the identification of obsolete items, which

should not be recommended anymore. This can be done both on the individual and community level. At the individual level, one could for example try to identify outliers in the user interest (e.g., when someone bought a gift for someone else in an online shop) or specific categories of items that the user has not interacted with for a longer period of time (e.g., an artist that was not played for years on a music streaming site by a certain user).

- *Repeated Recommendation:* In several application domains, recommending items that the user already knows, has consumed or purchased in the past can be meaningful. These application scenarios are not considered at all in the traditional matrix completion setup as described above. We can identify the following categories of repeated recommendation scenarios.
  - *Identifying repeated user behavior patterns.* Past interaction logs can be used by sequence-aware recommenders to identify patterns of repeated user behavior. A typical application example could be the repeated purchase of consumables, like printer ink. Such patterns can be both mined from the behavior of individual users, as in [149, 150], or the community as a whole. Another interesting field of application is App recommendation, in which frequent App usage patterns can be mined from user activity and used to enhance the user experience [7, 84, 99]
  - *Repeated recommendation as reminders.* In a different scenario, repeated recommendations can help to *remind* users of things they were interested in the past. Depending on the domain, these reminders could relate to objects that the user has potentially forgotten (e.g., an artist that she or he liked in the past), or to objects that the user has recently interacted with. The latter scenario is particularly relevant in e-commerce.

  In both mentioned scenarios, besides the selection of items to repeatedly recommend, a sequence-aware recommender has to reason about the *timing* of the recommendations [149, 150]. In the reminding scenario in e-commerce, the time frame to remind users of previously seen items might be narrow and objects may become obsolete very soon, e.g., if they were not purchased after a few view events. Nonetheless, always reminding users of reminding items they have inspected in the last session might be inappropriate if the user's current shopping intent does not match that of the previous session.

In the context of the recommendation of consumables, items can be repeatedly recommended after longer periods of times, e.g., weeks or even months. Proper timing can however still be important and such types of repeated recommendations share similar problems as *proactive* recommenders, e.g., that their recommendations might interrupt the user at the wrong time.

2.3.3 Order constraints

In certain domains, the characteristics of items may pose a sort of ordering between interactions. For example, it is reasonable to think that users that have watched a certain movie will watch its sequels later on, or a user who bought a printer will probably buy its replacement cartridges anytime soon.

We distinguish between *strict* and *weak* order constraints, and order constrains that *mined* directly from the logs of users' activity.

- *Strict or weak order constraints:* In the domain of the recommendation of a sequence of learning courses, for example, there might be strict requirements regarding the order of different courses that have to be considered by the recommender. e.g., when one cannot attend one course before another one was completed [144]. In the domain of movie recommendation, in contrast, it might be reasonable to recommend a sequel to a movie only after a user has watched the preceding episode. Such a constraint is however not necessarily strict. Weak order constraints are typical of music listening, in which the sequence of songs played next typically shares some musicological and style features with the previously played songs (e.g., genre, timbre, rhythm, etc.). This musicological 'coherence' constraints [67] restrict the pool of candidate songs in the recommendation list (or playlist) but does not pose any strict ordering between them.

- *Mined order constraints:* A sequence-aware recommender can mine such sequential patterns from the log data, e.g., to automatically detect that users who watched a certain movie later on have watched its sequel. Sequential pattern mining techniques in fact have been applied in the past in different application domains of recommenders, e.g., for predicting the next navigation actions of users on websites [95, 98] or the find next tracks to play in music recommendation problems [18].

2.4 Categorization of the state-of-the-art

We categorize the state-of-the-art of sequence-aware recommender systems with respect to 3 dimensions: the type of *context-adaptation*, the kind of *order constraints*, the usage of *item features* and the *application domain*. We believe that this categorization allows to highlight the overlooked aspects of this field and to accurately position this thesis wrt. the existing works.

Table 2.1 summarizes the categorization. Some considerations on the existing literature of sequence-aware recommender systems follows.

- *Context-adaptation:* From the first glimpse it is immediately evident the centrality of context-adaptation in sequence-aware recommendation, since almost the totality of the papers in the literature address this aspect, even if from different perspectives. More than half of the papers in the literature address the most basic form of context-adaptation based on the last-N interactions of the user. However, the interaction of the user with many existing online services is usually organized into sessions. By focussing only on the last-N interactions of the user we can oversimplify the problem and, for example, discarding relevant information at the beginning of a session, or discarding relevant informations contained in the past sessions of the user. For this reason, session-based and session-aware recommendation has gained significant traction in the recent years, also favored by new publicly available datasets (like the *30Music* dataset described in Chapter 8 and the XING dataset [1]) and the increasing interest from industry.

  While session-based recommendation has been addressed in relatively many existing works in the literature ($\sim 30\%$ of the papers analyzed), session-aware recommendation has received attention by a minority of the existing works. A possible reason can be found in the inherent complexity of modeling and combining the long-term and short-term interests of the user into a single effective recommendation model. We will address this point in Chapter 5, where we will present the details of a new personalized session-based recommender for returning users based on Hierarchical RNNs.

- *Subtask:* As we can see, only a minority of the existing works directly addresses one of the subtasks described before, with the recommendation of the most suitable next item or basket being the most investigated aspects. In this thesis we do not explicitly focus on any of these subtasks, but we envision their importance as future research directions for this field.

- *Order constraints:* The vast majority ($> 77\%$) of the research in sequence-aware recommendation focuses on patterns that are automatically mined from user log data. This is reasonable since the explicit definition of order constraints is not always possible and it frequently requires the adoption of costly procedures such as constraint satisfaction (see e.g. [101] or the addition of ad-hoc post-filtering stages to traditional item-to-item recommendation [67].

  This does not necessarily means that all sequence-aware recommenders adopt frequent pattern mining (FPM) techniques or its surrogates. In fact, as we will see in the next chapter, FPM-based recommenders represent only a fraction of the algorithms for sequence-aware recommendation. Many other techniques, such as Markov Models and Recurrent Neural Network, can implicitly represent order constraints between items by using sophisticate transition models.

  In the music domain, a question remains open: *does the order of the songs in the playlist matters in building good quality playlists?* Several works in the literature of playlist recommendation tried to answer this question from different perspectives (see [18] for a comprehensive review). In this thesis, we present a possible answer through an experiment based on randomized playlists and sequence-aware recommenders, that is presented in detail in Chapter 7.

- *Item features:* Surprisingly, the usage of item features and metadata has been completely neglected by the existing literature of sequence-aware recommenders. We show how rich item features can be effectively used in sequence-aware recommendation in Chapter 4.

- *Domains:* Some interesting considerations can be done on the application domains as well. From our analysis, most of the research in sequence-aware recommendation focuses on the e-commerce ($\sim 30\%$) and music ($\sim 26\%$) domains, followed by web-page ($\sim 13\%$) and Point-of-Interest ($\sim 10\%$). Other domains like advertisement, app and query recommendation are less represented. A reason for this can be certainly found in the larger number of publicly available datasets for the former two domains (see Table 2.2). This also serves as an indicator of the importance of sequential user feedback in every domain and on the quality of the sequence-based patterns that can be extracted from the logs of user.

  In this thesis, we had the opportunity to work on a variety of different domains. In Chapter 4 we will show experiments in video recom-

mendation and classified advertisement; in Chapter 5 we focus on job recommendation and, again, on video recommendation; in Chapters 6, 7 and 8 we instead show some novel developments in music recommendation.

Paper	Year	Type	Sub-task	Repet.	Trend	Ord.	Domain
Baezayates et al., [7]	2015	SA	NEXT			I	APP
Chen et al., [24]	2012	LI				W/I	MUS
Chen et al., [25]	2013	LI				W/I	MUS
Cheng et al., [26]	2013	LI				I	POI
Feng et al., [45]	2015	LI				I	POI
Garcin et al., [48]	2013	SB				I	NEWS
Grbovic et al., [53]	2015	LI				I	EC
Hariri et al., [54]	2012	SB				W/I	MUS
He et al., [57]	2009	SB	NEXT			I	QU
Hidasi et al., [61]	2016	SB				I	EC
Hosseinzadeh et al., [65]	2015	LI				I	MUS
Jannach et al., [66]	2015	SA				I	EC
Jannach et al., [67]	2015	SA	SIM/LIST			W/I	MUS
Letham et al., [76]	2013	LI				I	EC
Lian et al., [77]	2013	LI	BASKET			I	POI
Lim et al., [78]	2015	LI				S	POI
Lu et al., [84]	2014	SB	NEXT			I	APP
Maillet et al., [86]	2009	N/A	LIST			W	MUS
Mcfee et al., [90]	2011	LI				W/I	MUS
Mobasher et al., [95]	2002	SB	NEXT			I	PAGE
Moling et al., [96]	2012	SB					MUS
Moore et al., [97]	2013	LI	SIM		I	W/I	MUS
Nakagawa et al., [98]	2003	SB	NEXT			I	PAGE
Natarajan et al., [99]	2013	SA	NEXT			I	MUS/APP
Pauws et al., [101]	2006	N/A				S	MUS
Rendle et al., [112]	2010	LI	BASKET			I	EC
Rudin et al., [114]	2011	LI				I	EC
Shani et al., [123]	2005	LI				I	EC/PAGE
Song et al., [126]	2015	SB	NEXT			I	EC/VID
Sordoni et al., [127]	2015	SB	NEXT			I	QU
Tagami et al., [132]	2015	LI	BASKET			I	ADS
Tavakol et al., [133]	2014	SB				I	EC
Wang et al., [139]	2015	LI	BASKET			I	EC
Wu et al., [142]	2013	SB				W/I	MUS
Xiang et al., [143]	2010	SA				I	OTHER
Yap et al., [145]	2012	LI				I	PAGE
Yu et al., [146]	2012	LI	LIST			S	OTHER
Zhang et al., [148]	2013	LI	NEXT			I	ADS
Zhao et al., [149]	2012	LI		TIME		I	EC
Zhao et al., [150]	2014	LI		TIME		I	EC
Zhou et al., [151]	2004	LI	NEXT			I	PAGE
Zidmars et al., [152]	2001	LI				I	PAGE

Type SA: Session-aware, SB: Session-based, LI: Last-N interactions, N/A: no context-adaptation; **Sub-task** SIM: Find similar items, LIST: List continuation, BASKET: Basket recommendation, NEXT: "Logically" next item recommendation; **Repeated Recommendation** REP: Find items for repeated recommendation, TIME: Find timing for repeated recommendation; **Trend Detection** I: Detect individual trends, C: Detect community trend, S: Detect seasonal trends; **Application domain** APP: App recommendation, MUS: Music domain, QU: Query recommendation, EC: E-Commerce domain, CS: Learning courses recommendations, AD: Advertisements, PAGE: Web page recommendation, OT: Others; **Trend Detection** W: Weak, S: Strict, I: Inferred

Table 2.1: *Categorization of works on context-adaptation.*

Short name	Domain	Users	Items	Interactions	Sessions	Reference
Recsys Chall. 2015	E-commerce	N/A	38k	34M	9.5M	[61]
Ta-feng	E-commerce	32k	24k	829k	N/A	[139]
T-Mall	E-commerce	1k	10k	5k	N/A	[139]
AVITO	E-commerce	N/A	4k	767k	32k	[135]
Microsoft	Page	27k/13k	8k	13k/55k	32k/5k	[151, 152]
MSNBC	Page	1.3M/87k	1k	476k/180k	N/A	[145, 152]
Delicious	Page	8.8k	3.3k	60k	45k	[143]
CiteULike	Page	53k	1.8k	2.1M	40k	[143]
AOL	Query	650k	N/A	17M	2.5M	[127]
Foursquare 2	POI	225k	N/A	22.5M	N/A	[26]
Gowalla	POI	54k	367k	4M	N/A	[26]
Art Of the Mix	Music	N/A	218k	N/A	29k	[54, 67, 90]
XING	Job	785k	1M	8.8k	N/A	[1]

Table 2.2: *Publicly available datasets for sequence-aware recommendation.*

2.5 Algorithms for sequence-aware recommendation

The main computational task of any sequence-aware recommender system, as briefly introduced in the previous sections, is the automated extraction of sequence-related features and patterns from the logs of user actions.

The vast majority of the papers in the literature rely on sequence learning methods, such as sequential pattern mining and sequence modeling. This methods can deal with sequences of user actions naturally, and can generate recommendations tailored on the short-term activity of the user in a straightforward way. For example, frequent sequential patterns can be extracted from historical user sessions and collected in a database that, at recommendation time, is queried with the sequence actions performed by the user the current sessions, and the best matches are used to generate recommendations on what items the user should interact with next [95, 98].

While patter mining-based models decouple pattern extraction from recommendation, more sophisticated sequence modeling techniques, such as Markov models [24, 123] and, more recently, Recurrent Neural Networks [61], use advanced state-transition models learned from user logs to tackle both tasks at once. For example, in Markov Chain-based models [123], the transition probabilities are estimated on the seen transitions between items in user sessions. At recommendation time, the transition model is used to evolve the state of the user according to her most recent actions and to recommend the next actions proportionally to their likelihood to come next.

Class	Sub-class	Family	Papers
Sequence Learning	Frequent Pattern Mining	Frequent items & item-to-item	[66, 95, 98]
		Frequent sequences	[84, 95, 98, 114, 145, 151]
	Sequence modeling	Markov Models	[48, 57, 65, 90, 96, 123, 133]
		Recurrent Neural Networks	[61, 127, 148]
	Distributed item representations	Latent Markov embeddings	[24, 25, 45, 142]
		Distributional embeddings	[7, 39, 53, 132]
	Supervised models w/ sliding window		[7, 139, 152]
Matrix completion			[146, 149, 150]
Hybrid methods	Factorized Markov Chains		[26, 77, 112]
	LDA/Clustering w/ sequence learning		[54, 99, 126]
Others	Graph-based		[143]
	Discrete optimization		[67, 78, 101]

Table 2.3: *Taxonomy of algorithms for Sequential Recommender Systems.*

Beside sequence learning methods, sequence-aware recommendation can be framed as a traditional matrix-completion; however, straigth matrix-completion based on matrix and tensor factorization quickly becomes unmanageable even when modeling realitevly short user histories, and it is therefore used only in some borderline cases [149, 150]. More frequently, matrix-completion is combined with Markov Chains to build more flexible sequence-aware recommenders with higher robustness to data sparsity [26, 112].

In this section, we provide an overview of the algorithms for sequence-aware recommender. We identify three main classes of algorithms: *sequence learning*, *matrix-completion* and *hybrids*. We categorize the works in the state-of-the-art according to our taxonomy in Table 2.3.

2.5.1 Sequence Learning

Sequence Learning methods are used in a variety of application where data has an inherent sequential nature, like in natural language processing, time-series prediction and DNA modeling. Given the sequential nature of user actions, sequence learning methods are widely employed in sequence-

aware recommendation as well.

Frequent Pattern Mining (FPM)

Definition Frequent Patter Mining (FPM) methods were originally developed for discovering patterns in user consumption from large transaction databases [3, 4]. Patterns can be categorized into *unordered* patterns (or *itemsets*) and *sequential* patterns.

FPM extracts patterns of items that frequently occur together in the logs of user actions, e.g within sessions. Patterns having a *support* (i.e., the number of occurrences of the pattern in the dataset) greater than a predefined minimum support threshold are stored into a pattern database. Pattern databases usually rely on efficient data structures such as prefix-trees for fast query [95, 145]. At recommendation time, the recent actions of the user are mapped against the pattern database to generate recommendation lists.

Frequent Itemset and Association Rules (AR) mining [3] count the support of itemsets regardless of the order of the items within them. On the other hand, in Sequential Pattern Mining [4] the order of items is maintained. Contiguous Sequential Patterns enforces patterns to be formed exclusively by adjacent sequences of items.

SP mining is clearly preferable in sequence-aware recommendation since the order of the user actions is preserved and used to generated recommendations. This can be extremely beneficial in contexts in which the order of user actions is highly informative on her intents, such as in query [151] and app recommendation [84]. However, the strict sequentiality constraint of SPM can severely limit the number (and the quality) of the patterns that can be mined, especially in conditions of high data sparsity that is typical in recommender system scenarios.

Albeit less powerful, AR mining is still a valuable recommendation strategy for sequence-aware recommendation, even if the the order of the user actions is not maintained. Frequent itemsets are still extracted from sequences of user actions, such as user sessions, and the absence of the order constraint mitigates the impact of data sparsity. This turns out to be beneficial in certain sequence-aware recommendation scenarios such as page recommendation [95].

One extreme case of AR mining are simple item-to-item co-occurrence models [119], i.e. models that can generate suggestions of the type “users who bought this... also bought this...”, which are strictly with AR of size 2. Despite their simplicity, co-occurrence models were employed with great success in e-commerce [79] and music recommendation [66].

Existing works Mobasher et al. [95] and Nakagawa et al. [98] investigate the impact of sequential and non-sequential patterns in page prefetching and recommendation. Web usage data is first split into sessions, and then ARs, SPs and CSPs are extracted from sessions. In recommendation, the engine matches the current session of the user against the collection of rules. Candidates are generated using fixed-sized sliding window over the current session: the last N user actions are matched against rules of size $N+1$, and candidates get weighted by the confidence of the corresponding rules. To speed up the search, frequent patterns are stored into a directed acyclic graph (the Frequent Itemsets Graph). The authors show that less constrained patterns (AR and SP) have better quality in the page recommendation task, while CSP provide better performance in the page prefetching task. A very similar approach is described in [151].

With FPM personalization is obtained only by matching the activity of the user over a database a pre-extracted patters. Yap et al. [145] propose instead to weight patterns according the their personalized relevance for the user. They compare non-personalized popularity-based weighting schema against three personalized ones (cosine similarity, Longest Common Subsequence and Competence Score). Specifically, the Competence Score measures the extent to which a candidate pattern is compatible with the sequence of interactions in the current user session and it can be effectively used to recommend more next-items for the target user. Personalized weighting schemes outperform non-weighed sequential patterns and non-personalized weighing schemes.

Finally, Lu et al. [84] present Mobile Application Sequential Patterns (MASPs) mining for usage prediction in smart-phones. Transactions are composed by sequences of apps together with the location of the user at each time. The MASP-mine algorithm takes into account both user movements and Apps launched to discover complete mobile App sequential patterns. At prediction time, the pattern with maximal support that matches the recent user movements and activity is used to predict the App launched next.

The main drawback of FPM methods is their sensitivity to the minimum support threshold used to distinguish frequent patterns from infrequent ones. Weak minimum support thresholds will return a huge number of noisy rules, whereas strict minimum thresholds may discard important rules that are rare. Rudin et al. [114] propose to use an "adjusted" confidence score to address these limitations. Alternatively, Mobasher et al. [95] and Nakagawa et al. [98] propose an *all-kth-order* method to increase the flexibility of the model without resorting to too weak minimum support

thresholds. For a given collection of rules extracted with a fixed minimum-support threshold, the recommendation engine first uses the largest possible active session and then iteratively decreases its size until a recommendation is generated. The algorithm therefore looks for higher quality rules first and then reduces its quality constraint until a rule is found. Zhou et al. [151] use a very similar approach by iteratively removing the first item in the current usage sequence until a matching pattern is found.

Sequence modeling

Description Every sequence of user's interactions is to all effects a time series with the discrete observations and optional time-stamps attached, and can be therefore studied with complex time-series models. However, since the role of time-stamps in sequence-aware recommendation is principally used to sort user's actions and it is not used to the end of the recommendations the time variable can be safely ignored and "simpler" sequence models used instead.

In general, sequence modeling (and, obviously, time-series) allows to model the effect of past observations over future ones. In sequential recommendation, this means that we can model the impact of past user interactions over her future ones, and use these models to generate recommendations from her recent activity. Sequence modeling methods for in sequence-aware recommendation mainly belong to two categories: Markov Models and Recurrent Neural Networks (see Table 2.3).

Markov Models Markov models represent sequential data as stochastic process over discrete random variables (or states). The Markov property limits the dependencies of the process to a finite history. For example, in first-order Markov Chains (MCs) the transition probability of every state depends only on the previous state. Higher-order MCs use longer temporal dependencies to model more complex relationships between states [2].

In sequence-aware recommender systems, the Markov property translates into assuming that the item(s) the next user action depends only on a fixed number of her most recent actions. This assumption is consistent with the intuition that the near history often is more relevant than the far past, but severely limits the possibilities of the model to represent long-term dependencies. However, the Markov assumption is necessary to make the space of possible transitions manageable from the model, an issue that affects specially Markov Chains.

[2] See [46] for details on Markov models for pattern recognition

In fact, MCs cannot be naïvely applied to sequence-aware recommendation since data sparsity quickly leads to poor estimates of the transition matrices.
Shani et al. [123] enhances Markov Chains with several heuristics – namely skipping, clustering and finite mixture modeling – to mitigate the impacts of data sparsity. The model is trained over data coming from the purchase records of the users of an online book store, and shows the superiority of sequential models over non-sequential ones. McFee et al. [90] use Markov Chain mixtures for playlist generation. The mixture is a weighed ensemble of several MCs (uniform, weighted and k-nearest neighbors) whose weights are learned via maximum-likelihood optimization on a training dataset of playlists.

Shani et al. [123] and Moling et al. [96] use reinforcement-learning based on Markov Decision Processes (MDPs) for sequence-aware recommendation. User decisions are modeled as sequential decision making problem – backed by an underlying MC – which allows to tailor recommendations not only on the recent user activity but also on the expected reward (income) for the shop. As for MCs, the state space of MDPs quickly becomes unmanageable in real-life scenarios. In Tavakol et al. [133] the state space is factorized over a set of mutually independent item attributes to reduce complexity. In the case of a clothing marketplace, for example, dress characteristics such as category, color, gender and price can be considered; the sequential relationships between attributes are independently modeled by the MDP to predict the characteristics of the products the user will likely search next.

Another strong limitation of MCs is the absence of an universally good value for the order of the model. He at al. [57] use a mixture of Variable-order Markov Models (a.k.a. context trees), which use a context-dependent order to capture both large and small Markov dependencies in the data, for session-based query recommendation [3]. A different approach was adopted by Garcin et al. [48] in the context of a news recommendation application. They assign one predictor (expert) to each context (node) in the context-tree, being each node associated with a different order of the Markov model. Each predictor is then trained to predict the forthcoming news article given its context. As the sequence of actions of the user grows, deeper nodes in the tree become active and contribute to the final recommendations.

The limitations of MCs are partially addressed by hidden-state models, like Hidden Markov Models (HMMs). In HMMs, each hidden (or latent) state is a discrete variable associated with a probability distribution over the

[3] See [10] for a comprehensive analysis of algorithms for learning VMMs.

observed variables. In conformity with the Markov property, every hidden state in the HMM is conditionally dependent only on the previous one [49]. HMMs for sequence-aware recommendation model user actions as observations of some discrete hidden state whose distribution changes with the changes in the user's behavior. Therefore, the hidden states of the model are a possible representation of the context of the user [65]. However, their discrete-valued hidden states poses strong constraints on the contextual information that they can store, which in turn limits the applicability of HMMs to sequence-aware recommendation.

Recurrent Neural Networks Recurrent Neural Networks (RNNs) [64] are distributed real-valued hidden state models with non-linear dynamics [4]. At each time step, the current input in the sequence and the hidden state from the last step are used to update the hidden state of the RNN. The hidden state is then used to predict the probability of the forthcoming items in the sequence. The recurrent feedback mechanism memorizes the influence of each past data sample in the hidden state of the RNN, hence overcoming the fundamental limitation of Markov Chains. They are therefore good candidates for modeling the complex dynamics in user action sequences. Variants of RNN such as LSTM [52] and GRU [28], by means of their sophisticate hidden dynamics, can model much longer and complex temporal dependencies than other competitive approaches, like Hidden Markov Models.

Zhang et al. [148] use RNNs for sequential click prediction in advertisement. At each time step, the RNN is trained to predict the next click of the user given her current click and the previous state of the network using a classification loss (cross-entropy).

Hidasi et al. [61] use Gated Recurrent Units (GRUs) [28] – a variant of RNNs – for modeling user activities in a session-based scenario. The model is still trained to predict the next item in the sequence given the current one, but the addition with a sampled pairwise ranking loss functions such as BPR [111]. This approach is extremely effective in a pure session-based scenario, without any historical information, and outperforms item-to-item models, the industrial state-of-the-art approach for large scale session-based recommendation. We will describe this model in detail, and propose further development to it, in Chapters 4 and 5.

Sordoni et al. [127] propose a generative model for session-based query recommendation based on Hierarchical Recurrent Encoder-Decoder RNN. The next query recommended to the user is generated directly from a rep-

[4] See [80] for a comprehensive review on Recurrent Neural Networks.

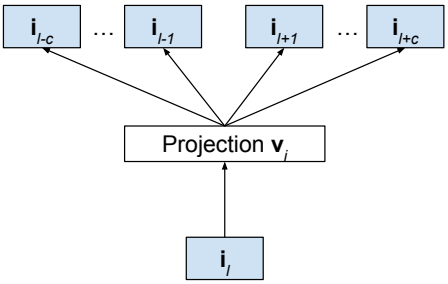


Figure 2.2: *Distributed item representations for sequential recommendation with Prod2Vec skip-gram [53].*

resentation of the text of the previous query (the first level of the hierarchy) and a representation of all the previous queries issued by the user in the current session (the second level of the hierarchy).

Distributed item representations

Description Distributed item representations are dense, lower-dimensional representations of the items built out of sequences that can preserve the sequential relationships between items. Similar to latent factor models, every item is associated with a real-valued *embedding* vector, which represents the projection of every item on a low-dimension space in which certain item transition properties are preserved. Given the current session of the user, or the sequence of her most recent actions, recommendations on the next-item(s) are generated by traversing the embedding space in a stochastic fashion [24], or by searching the nearest neighbors to the last item(s) explored by the user [53].

Existing works Chen et al. [24] present the Latent Markov Embedding (LME), a regularized maximum-likelihood embedding of Markov Chains in Euclidean space. Every item is projected into a space such as the distance between any pair of items in this space is proportional to their transition probability in a first-order Markov Chain. The resulting vector spaces can be traversed to generate sequences stochastically (e.g., playlists). Chen et al. [25] extend LME by clustering items and adding cluster-level embeddings to account for locality in item transitions. Wu et al. [142] propose Personalized LME (PME) where both items and users are projected into a Euclidean space so that the strength of their relationship is reflected by the

projection. This allows to incorporate long-term user preferences into the model and to use them in recommendation. Feng et al. [45] use a pairwise ranking function analogous to the Bayesian Pairwise Ranking [111] to condition the transition probabilities on the personalized ranking preferences of users.

Another approach consists in leveraging the so-called "distributional hypothesis". In linguistics, this hypothesis states that semantically equivalent words occur frequently in the same contexts. This hypothesis allows to generate distributed representations of words that preserve their syntactical and semantical properties, like Word2Vec [93] and GloVe [102]. Similar considerations apply to sequence-aware recommendation too. Grbovic et al. [53] propose Prod2Vec, a SRS based on distributed product representations learned from sequences of e-mail receipts. Similar to LME, every item is projected onto a lower dimension space in which such hypothesis is preserved. Specifically, Prod2Vec uses the skip-gram model (shown in Figure 2.2), which projects items that tend to have similar neighboring items (i.e., items that tend to be 'surrounded' by the same set of items) close one to the other. Once learned the item vectors, the recommendations for the last-N item in the user sequence is given by the top-K most similar items in the resulting vector space computed with cosine similarity, discounted with a temporal decay factor to account for aging effects. Additionally, when users can interact with multiple items at the time, the skip-gram model can be extended to take sets of items in input [53]. In order to account for the directed order dependencies between interactions, the skip-gram model can be modified to consider only future items as context [39].

Supervised learning with sliding windows

Description While sequence learning methods rely on ad-hoc solutions to model sequences, sliding window models convert sequence prediction into a traditional supervised learning problem that can be solved with any off-the-shelf classifier such as decision trees, feed-forward neural networks and learning-to-rank methods.

The approach closely resembles autoregressive models and it is relatively simple. A sliding window of size W is passed over each sequence (see Figure 2.3). At each step, all the items within the window are used to build the features of the supervised learning problem and the identifier of the immediately next item is used as target variable. Consequently, the sequence prediction problem is converted into a multi-class classification problem, or into a multi-label classification if multiple target items are al-

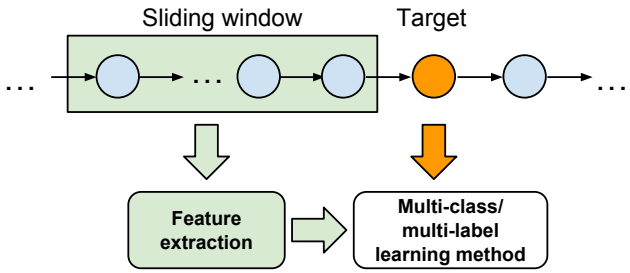


Figure 2.3: *Sequential recommendation with generic supervised learning with sliding window.*

lowed.

However, this approach has several shortcomings. First of all, it is hard to define the descriptive features properly, since they can be heavily domain dependent and can be hardly reused. Secondly, results can be highly sensitive on the choice of W. Finally, setting up a multi-class (or multi-label) classification problem can be extremely computationally expensive when the set of items grows considerably, and the quality of the learned model can be severely affected by unbalanced distributions in the target variable, which are common in real-life recommendation scenarios. For these reasons, ad-hoc sequence models are usually preferred over supervised models with sliding window.

Existing works Nevertheless, there are few interesting applications of this approach to sequence-aware recommendation in the literature. Zidmars et al. [152] frame sequential click prediction as a binary prediction problem. Click streams data is "expanded" by defining a set of 'accumulator' variables (*lagged* and *cache* variables) to represent the contextual and historical activity of the user. Finally, a page-level probabilistic decision-tree model is trained classify requests on the next page; at recommendation time, pages are ranked according to the predicted probability of being the next page.

Beaza Yates et al. [7] use contextual features extracted from the app usage log of the last 12 hours for sequential app usage prediction. These contextual features include some *basic* features, such as geo-location and phone usage features, and *session* features, which are basically the Word2Vec representations [93] of the actions of the user in the sliding window. Finally, they use a parallelized version of the Tree Augmented Naive Bayes (TAN) algorithm for the classification of the next app usage.

Wang et al. [139] use Feed-forward Neural Networks for next-basket recommendation where only the items in the previous basket are used to rank the items in the next-one ($W = 1$). The first layer of the NN encodes the previous basket as fixed-size real-valued feature vector that is computed by taking the element-wise maximum (*max-pooling*) or average (*mean-pooling*) of the embeddings of the items that compose the basket. The second and final layer of the NN learns to rank the items in the next-basket by taking as input the concatenation of the user's and previous basket's embeddings. All the embeddings are learned end-to-end by means of a ranking loss function.

2.5.2 Matrix completion

Sequence-aware recommendation, as we have already discussed, fundamentally differs from traditional matrix-completion. However, there are a few cases in which this paradigm can be applied. also to sequence-aware recommendation.

Zhao et al. [149, 150] solve a matrix-completion problem on the item purchase matrix. The model, called Purchase Interval Matrix Factorization (PIMF), factorizes the matrix by using a weighted loss function that maximizes the expected utility for the user, in which the utility is weighed on observed time intervals between each pair of items. The resulting model can predict the personalized relevance of an item depending on the time at which recommendations are generated, and hence providing an effective strategy for sequence-aware recommendation.

Personalized story generation is the application domain of the approach proposed by Yu et al. [146]. Stories are represented as a prefix graph, in which each node represents a prefix of a story, i.e. a possible sequence of point plots. In their problem encoding, they are given a "prefix-rating matrix" to be completed, where the items are replaced by possible story prefixes and users provide ratings only to some of such prefixes. Matrix factorization is used to predict the missing entries in the prefix-rating matrix. The highest scored full-story that descends from the current "story-so-far" of the user is used to suggest the next plot point. The rating of the user is collected on the next plot point, and the process continues iteratively (factorization, recommendation, rating) until the story reaches an end.

2.5.3 Hybrid methods

Description Hybrid models combine the flexibility of sequence learning methods with the robustness to data sparsity of factorization-based matrix-

completion techniques. Such hybridization enables sequence learning methods to use powerful learning-to-rank collaborative models (such as BPR [111]), which cannot be done in standard sequence learning, apart from few exceptions [61].

Existing works A first hybridization technique is to combine matrix factorization with Markov Chains trough tensor factorization. Rendle et al. [112] propose pairwise tensor factorization technique for next-item recommendation named Factored Personalized Markov Chain (FPMC). In the case of first-order MCs, user interactions can be represented as a 3-dimensional (*user*, *current item*, *next item*) tensor, in which each entry in the tensor is an observed transition between two items performed by a specific user. Since only a tiny fraction of the possible transition pairs are observed, the resulting tensor will be extremely sparse. FPMC uses pairwise factorization with BPR loss to predict the unobserved entries in the tensor, thus to predict personalized transitions between pairs of items. In short, FPMC can be seen as a first-order Markov Chain whose transition matrix is jointly factorized with a standard 2-dimensional user-item matrix factorization. The joint factorization allows to infer the unobserved transitions in the Markov Chain from other users' transition pairs. At recommendation time, items are ranked according to their likelihood to be the next item given the last item the user has interacted with.

Due to its robustness to data sparsity, FPMC has been used in differnt domains. Rendle et al. [112] use it for next-basket recommendation. Lian et al. [77] use FPMC directly in check-in group prediction.

Other types of hybrids exist. In the context of playlist recommendation, Hariri et al. [54] use Latent Dirichlet Allocation (LDA) [17] to extract latent topics from playlists, where playlists are taken as documents and songs as words for the LDA. Sequential pattern mining is applied on top to find patterns of latent topics in playlists. At recommendation time, the frequent patterns are mapped to the topics extracted from the listening session and used to filter the recommendations generated by a classical user-based KNN recommender.

Song et al. [126] propose the States Transition pair-wise Ranking Model (STAR), which combines LDA with first-order MC to simultaneously model the user's long and short-term favorites. User's long-term favorites are represented as topics generated by LDA with a user's specific prior. User's short term favorites are instead captured by a transition matrix in the MC. The unified model is basically an hidden Markov Model whose latent states are controlled by the personalized LDA generative model, and it is trained

via full Bayesian inference with Markov Chain Monte Carlo (MCMC).

Finally, Natarajan et al. [99] propose iConRank, a method based on behavioral clustering to introduce personalization into sequence models. Behavioral clustering groups users with similar sequential behavior by applying *k-means* clustering on the per-user transition matrices of a first-order MC. Personalized PageRank algorithm is then used to build transition models at cluster-level. At recommendation time, the user is mapped to its corresponding cluster and the cluster-level transition model is used to provide sequential recommendations.

2.5.4 Other methods

Only a handful of methods do not fit into the above categories.

Graph-based methods

Xiang et al. [143] use a graph-based approach, where long-term and short-term user's preferences are fused into a two-sided bipartite graph, the Session-based Temporal Graph (STG). One side of the STG connects users with the items they have ever interacted with (long-term preference). The other side of the graph instead connects session identifiers with the items the user has interacted with in the session (short-term preference). Edges are weighted to reflect influence of long and short-term preferences more precisely. The relationships between items are propagated through the graph via Injected Preference Fusion (IPF). At recommendation, the STG is traversed via the Temporal Personalized Random Walk (TPRW) to generate session-aware recommendations.

Discrete optimization

Discrete optimization is commonly employed in sequence-aware recommendation when weak or strict ordering constraints between items exist. Typical examples are playlist generation, travel planning and learning course sequence generation.

Pauws et al. [101] solves Automated Playlist Generation (APG) with local search. APG generates sequences of songs that satisfies a set of constraints, such as: (i) each pair of adjacent songs must be similar, (ii) each two songs must be sufficiently different and (iii) the playlist must have a maximum duration, and so on. This is an NP-hard Constraint Satisfaction Problem problem that is solved via local search with simulated annealing.

Lim et al. [78] use discrete optimization for personalized tour recommendation. Tours are represented by sequences of POI visits extracted

from geo-tagged photos. The historical activity of each user is first split into sequences of tours and used as ground truth. Then, the tour recommendation is solved as an integer programming problem that looks for the optimal sequence of POIs with a fixed start and ending location and a given time budget. The objective function combines different non-personalized scores, such as the popularity of each POI, and personalized scores, such as time-based and frequency-based user interests and personalized POI visit duration.

Beside sequence generation, Jannach et al. [66] use discrete optimization for next-song recommendation. Candidate songs are first scored by a complex session-to-item kNN procedure, which weights candidate songs according to their compatibility with the tracks recently listened by the user in terms of, e.g., playlist-level and song-level content-based similarities (e.g., based on tags and musicological features). To account for the weak 'coherency' constraints that exist between subsequent tracks in a listening session or in playlists, the list of top-N candidates is re-ranked at recommendation time though a greedy procedure that minimizes the difference between the sequence of songs and the recommended list wrt. a chosen group of (musicological or content) features, showing superior recommendation performance wrt. the unconstrained variants.

2.6 Evaluation of sequence-aware recommender systems

In this chapter, we present the evaluation procedures that can be used in the evaluation of sequence-aware recommenders. First, we analyze in detail offline evaluation protocols together with some practical guidelines for the evaluation of this family of RSs. We then provide a brief overview of online evaluation.

2.6.1 Offline evaluation

In offline evaluation, the quality of RS is measured without inquiring real users, which is usually a slow, intrusive and expensive procedure. The quality of the results obtained by an offline evaluation procedure depends strongly on the data partitioning procedure, on the evaluation protocol and on the metrics that are employed. The whole evaluation procedure must reflect as much as possible the real application scenario [32].

As a general guideline, the evaluation procedure must be carefully chosen according to the actual sequential recommendation scenario that we are modeling (next-item, sequence, session-based or session-aware sequential recommendation) and to the computational resources that are available.

We describe here the data partitioning techniques and the evaluation protocols that are used in the evaluation of sequence-aware recommender.

Data partitioning procedures

The first step of the evaluation of a generic RS is the generation of the training and test data-sets. The former is used to train the recommendation algorithm, whereas the second contains held-out data used exclusively during the evaluation. Additionally, a fraction of the data in the training dataset can be held-out for tuning the hyper-parameters of the recommendation algorithm (the validation set).

Due to the sequential nature of data, the dataset partitioning of sequence-aware recommender differs from classical item recommendation [59]. Dataset partitioning procedures can be classified into *between-sequence* and *within-sequence* procedures. Between-sequences procedures partition the dataset by holding out complete sequences, i.e., without breaking sequences apart. On the other hand, within-sequence procedures act directly over the individual interactions that compose the sequences. Each sequence can be split into contiguous sub-sequences that are assigned either to the training or to the test dataset.

Partitioning procedures can be additionally classified into *temporal* and *non-temporal* procedures. Temporal procedures consider the temporal ordering between interactions and sequences to partition the dataset. They can be used exclusively when all the sequences or the events in the dataset can be ordered in time (i.e., when all time-stamps are available). On the other hand, non-temporal procedures partition the dataset without considering time and can be with any sequence dataset.

Taxonomy We now present a taxonomy of the data partitioning procedures for sequential recommendation according to the aforementioned categories.

- *Split between-sequences*. The procedure assigns *complete* sequences to either the training dataset $\mathcal{S}_{\text{TRAIN}}$ or the test dataset $\mathcal{S}_{\text{TEST}}$. The procedure never separates the interactions within a sequence. The splitting can be performed in two different ways:
  - *Random hold-out* (Figure 2.4a). Sequences in $\mathcal{S}$ are randomly assigned to either one of the two datasets. The ratio between the size of the training and test datasets in terms of number of sequences is fixed a-priori (e.g., 60% training and 40% test, 60/40 in short), but the number of interactions contained in each partition can vary significantly with different sampling seeds. For this

reason, this evaluation procedure is usually combined with cross-validation in order to increase its significance and reliability.

- *Temporal hold-out* (Figure 2.4b). The dataset is partitioned according to their temporal ordering by using temporal threshold t_{TEST}. Sequences in $\mathcal{S}$ are first ordered by the time-stamp of their first interaction. Then the procedure assigns all the sequences starting before t_{TEST} to $\mathcal{S}_{\text{TRAIN}}$ and the sequences that start after to $\mathcal{S}_{\text{TEST}}$. The number of sequences and interactions in each partitions depends on the temporal distribution of the activity of users. Common choices are, for example, setting t_{TEST} equal to the last day in the dataset, or the last week or month in order to reduce the impact of weekly effects on the evaluation.

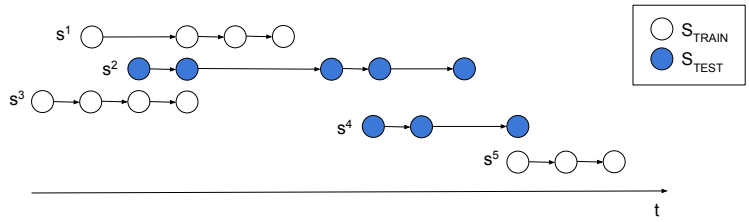
- *Split within-sequences*. The procedure partitions the interactions within each sequence into at most two contiguous subsequences that are assigned either to the training or to the test datasets.

  - *Leave last-k out* (Figure 2.4c). Assigns the last k interactions of a sequence to the $\mathcal{S}_{\text{TEST}}$ and the remaining part of the sequence to $\mathcal{S}_{\text{TRAIN}}$. All the sequences in $\mathcal{S}$ are affected by the split. The number of interactions in $\mathcal{S}_{\text{TEST}}$ is fixed and equal to $k|\mathcal{S}|$.
  - *Temporal hold-out* (Figure 2.4d). All the sequences having interactions before and after a temporal threshold t_{TEST} are split into two contiguous subsquences. The subsequence of interactions before t_{TEST} is assigned to $\mathcal{S}_{\text{TRAIN}}$. The remaining subsequence is assigned to $\mathcal{S}_{\text{TEST}}$. All the sequences ending before t_{TEST} are completely assigned to $\mathcal{S}_{\text{TRAIN}}$. All the sequences starting after t_{TEST} are completely assigned to $\mathcal{S}_{\text{TEST}}$.

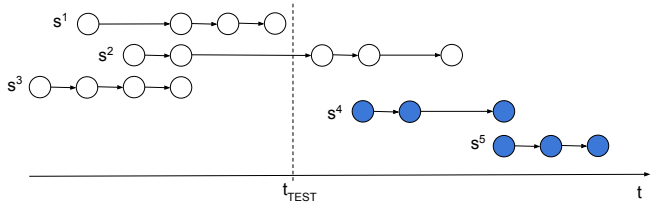
Discussion Each data partitioning procedure has its own advantages and drawbacks depending on the characteristics of the data (Table 2.4).

- *Temporally ordered data*. When the time-stamps are available, all the interactions in the dataset $\mathcal{S}$ can be ordered absolutely in time, as shown in the examples in Figure 2.4.

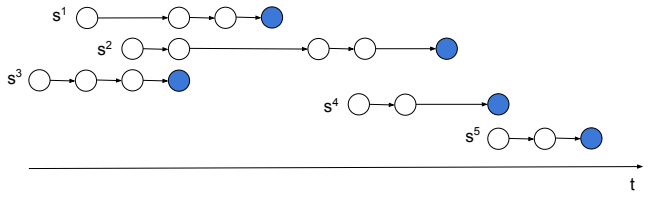
  - *Temporal splits* preserve the temporal ordering among the interactions in the training and test datasets to different extents. In between-sequence temporal hold-out, there is no sequence in the training dataset that starts after any sequence in the test dataset; however, it is still possible to have interactions in the training



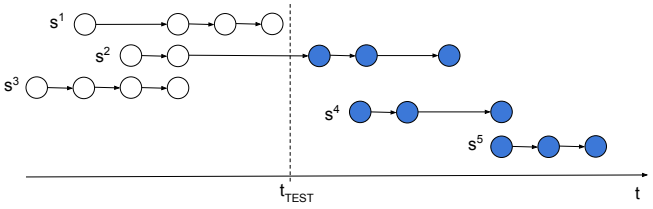
(a) *Between-sequence random hold-out*



(b) *Between-sequence temporal hold-out*



(c) *Within-sequence leave last-*1 *out*



(d) *Within-sequence temporal hold-out*

Figure 2.4: *Partitioning of a sequence dataset. Circles in white/blue represent the sequences and interactions assigned to the training/test datasets respectively.*

dataset that occurred after some of the interactions in the test dataset (Figure 2.4b). In within-sequence temporal hold-out, the temporal partitioning is performed at interaction level. For this reason, this procedure is the one that most strictly preserves the temporal ordering among interactions, because no interaction in the training dataset could occur after any interaction in the test dataset (Figure 2.4d).

  - *Non-temporal splits* partition the dataset without considering the temporal ordering between interactions. This can be source of undesired leakage of information from the training to the test phase. It is actually possible to partition the dataset in a way such that the training set contains interactions or complete sequences that occurred after the interactions in the test dataset. In the example in Figure 2.4a, the training sequence s^5 contains interactions that have been generated after all the interactions in the sequence s^2. Similarly, in Figure 2.4c the test interaction of sequence s^1 precedes all the training interactions of sequences s^4 and s^5. As a consequence, the recommendation system will be trained over data that was generated after the data used for testing. While this is feasible in offline evaluation, this clearly clashes with the characteristics of any real-life scenario, in which "future" data is clearly not available to train our models. This fact can severely limit the practical utility of the evaluation results achieved with this approach. For this reason, temporal split procedures should be generally preferred over non-temporal ones in datasets with temporally ordered data.

- *Non-temporally ordered data*. When time-stamps are not available, interactions can be ordered only relatively to each sequence. For example, in music playlists songs are ordered in the way they are played within each playlist, but there does not exist any absolute ordering between all the songs played in the playlist dataset.

  - *Temporal splits.* In these scenarios, temporal partitioning procedures cannot be used, and non-temporal procedures must be used instead.
  - *Non-temporal splits.* Random-hold out can be used multiple times over the same dataset with cross-validation to increase the significance of the evaluation results. However, the evaluation procedure runs over complete sequences, which can be a computationally expensive procedure depending on the length of the se-

Procedure		Pros	Cons
between sequence	sequence hold-out	cross-validation is possible	temporal leakage of information
	temporal hold-out	preserve sequence temporal order	time-stamps needed; test size depends on temporal data distribution
within sequence	leave last-k out	balanced testing	temporal leakage; biased towards the last interactions of each sequence
	temporal hold-out	preserves interaction temporal order	time-stamps needed; test size depends on temporal data distribution; not all sequences are split

Table 2.4: *Comparison of the partitioning procedures for sequence datasets.*

quences and on the nature of recommendation algorithm.
On the other hand, leave last-k out partitioning is generally less computationally expensive since only the last k terms of every sequence are used in the evaluation. In the extreme case, only the last interaction is considered by the evaluation procedure ($k = 1$). However, the main drawback of this procedure is that it tends to be biased towards the ending of the sequence of interactions. While this can be a desirable effect in application scenarios such as e-commerce where the last interactions usually corresponds to a conversion event, in other application scenarios it can be source of biased evaluation results. For example, in playlist recommendation, the last track in human generated playlists is frequently chosen to be the most popular one to create a sort of "grand finale" effect [18]. Hence, predicting the last track can be much easier than predicting the previous ones. A way of overcoming this problem is to consider more than a single item in the test set, i.e. using $k > 1$.

Evaluation protocols

We now describe the possible evaluation procedures. We consider that sequences in the sequence dataset has been partitioned into two datasets $\mathcal{S}_{\text{TRAIN}}$ and $\mathcal{S}_{\text{TEST}}$ with one of the aforementioned data partitioning proce-

Symbol	Meaning
$\mathcal{S}$	the sequence dataset
$\mathcal{S}_{\text{TRAIN}}$	the training dataset
$\mathcal{S}_{\text{TEST}}$	the test dataset
p	the user profile
q	the ground truth
x	a generic interaction
E	a set of interactions
L	a top-N recommendation list
d	the length of the look-ahead sequence
$q[:d]$	sub-sequence with the first d elements of q
$set(s)$	set of items in the sequence s

Table 2.5: *Notation used in the description of the evaluation procedures*

dures.

The evaluation protocols for sequence-aware recommenders must consider the sequential nature of the data in the test dataset, and the classical evaluation of top-N recommenders cannot be used directly with sequence-aware recommenders.

We identify 3 possible evaluation protocols: *sequential evaluation*, *sequential evaluation with look-ahead*, *set evaluation*.

Another fundamental assumption is the Missing-as-Negative (MAN) one. The MAN assumption, commonly used in the evaluation of recommendation systems, considers the unseen interactions are taken as negative examples [129]. Hence, the interactions in the test sequence are considered as the only relevant ones for the user. This assumption allows to the evaluate the quality of the next-item recommendation by means of accuracy metrics, such as Mean Average Precision and Recall. The evaluation metrics used in the evaluation of sequence-aware recommenders will be discussed in a dedicated subsection later on.

Protocols We now provide a concise description of the evaluation protocols for sequence-aware recommenders based on the MAN assumption. The notation is described in Table 2.5.

The evaluation of sequence-aware recommender requires to define of a user profile p, used to a recommendation list, and of a ground truth q, used to evaluate the quality of recommendations produced.

In the case of partitioning between-sequences, $\mathcal{S}_{\text{TEST}}$ is made of complete sequences. Every sequence $s \in \mathcal{S}_{\text{TEST}}$ can be partitioned into user profile and ground truth with one of the following two approaches:

- leave last-m out (LLO): put the last m interactions of s in the ground truth and the remaining interactions in the user profile

- keep first-l in (KFI): put the first l interactions in the user profile and the remaining in the ground truth

Both approaches split each test sequence s into two contiguous sub-sequences p and q. In LLO, the length of the ground truth q is fixed a-priori for every sequence in $\mathcal{S}_{\text{TEST}}$, but the length of the user profile p can vary, with that the information available to the sequence-aware recommender at recommendation time. Conversely, with KFI the length of the user profile is fixed, but the number of interactions in the ground truth depends on the length of the test sequence s.

Conversely, on the case of partitioning within-sequences, each sequence $s \in \mathcal{S}$ has already been split into user profile p and ground truth q (excluded the borderline cases in which individual sequences are not split, e.g., when using temporal thresholds to partition the dataset).

Notice that, when multiple types of interactions are available (e.g. clicking, bookmarking or replying to a job posting), the ground truth q can be restricted to the interactions of a given type (or types) of interest.

Once the user profile p and the ground truth q have been defined, then the following procedures are available to evaluate the recommendation quality of sequence-aware recommender under examination.

- *Sequential evaluation.*

  i) Given p, generate a top-N recommendation list L;
  ii) Evaluate L over the *first* interaction $x \in q$;
  iii) Append x to p and remove it from the head of q;
  iv) Repeat (i) until q is empty.

- *Sequential evaluation with look-ahead.*

  i) Given p, generate a top-N recommendation list L;
  ii) Evaluate L over the *set of the first* d interactions $E = set(q[:d])$;
  iii) Append $q[:d]$ to p, and remove it from the head of q;
  iv) Repeat (i) until q is empty.

- *Set evaluation.*

  i) Given p, generate a top-N recommendation list L;
  ii) Evaluate L over the *set* of interactions $E = set(q)$.

Discussion Each evaluation protocol provides different estimates of the sequential recommendation quality of a sequence-aware recommenders. Therefore, two different sequence-aware recommenders must be compared under the same evaluation protocol to obtain a fair comparison. Additionally, the evaluation protocols can significantly differ in the number of recommendation requests. Hence, the evaluation protocol must be carefully chosen according to the computational resources that are available.

- *Sequential evaluation* simulates interactive evaluation in which the actions in the ground truth are considered as the optimal ones. By preserving the original order of the interactions in the ground truth, it allows to evaluate the behavior of the sequence-aware recommender with a dynamically evolving user's history. Additionally, it allows to evaluate how the sequence-aware recommender behaves when new interactions are sequentially added to the user's history. For these reasons, this is the preferred choice for the evaluation of next-item recommender systems. However, it can be very computationally demanding since it requires to generate and evaluate a number of recommendation lists equal to the number of interactions in the ground truth.

  It is important to notice that in sequential evaluation is based on the strong sequencing assumption that the next interaction is the only positive one. This strong sequencing assumption that may be too restrictive for domains with weak-ordering relationships between items, such as, for example, music, and underestimate the actual perceived quality of recommendations.

- *Sequential evaluation with look-ahead* is a relaxed version of sequential evaluation in which recommendations are evaluated over a *subset* of forthcoming interactions, the so-called look-ahead set. The look-ahead can be defined by taking a fixed-number of the next interactions in the ground truth. Another option is to define the look-ahead as the set of interactions generated in a fixed time-interval, such as, for example, all the interactions generated in the next hour or day.

  In the evaluation of a recommendation list L, all the items in the look-ahead set E are considered as true positives. This relaxes the sequencing assumption of sequential evaluation, since the interactions in the look-ahead set can occur in any order. The sequence of interactions in the look-ahead is then concatenated to the user profile in its original order, and the evaluation proceeds with the following look-ahead set. This evaluation protocol is less computationally expensive than

sequential evaluation, since the ground truth is split in sets of roughly equal size or equal time-span. By means of a look-ahead set, this procedure relaxes the strict sequencing assumption used in sequential evaluation while partially retaining the sequential behavior of the user. However, the evaluation results can be highly sensitive on choice of the look-ahead set [53].

- *Set evaluation* completely brakes the ordering of the interactions in the ground truth. All the interactions in the ground truth are considered as true positives, regardless the order in which they were actually performed by the user. Hence, this evaluation procedure allows to measure the capability of the sequence-aware recommender to predict interactions the user will *eventually* perform. This evaluation protocol is therefore significantly less computationally demanding than sequential evaluation, since it requires to generate a number of recommendation lists equal to the number of sequences in test set. However, it completely disregards the sequential order in the interactions in the ground truth, which can lead to misleading results on the quality of the recommendation issued by the sequence-aware recommender.

Evaluation metrics

All the evaluation protocols require to evaluate a recommendation list L over a set of items, where this set is composed of a fraction of the items in the ground truth. This is a classical top-N recommendation scenario [32] that can be evaluated with *accuracy* metrics. The values of the accuracy metrics are commonly averaged at the end of the evaluation to obtain a single evaluation score. A non-exhaustive list of the evaluation metrics that can be used for top-N recommendation is Precision, Recall, Mean Average Rank (MAR), Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), Normalized Discounted Cumulative Gain (NDCG) and the F1 metric.

Evaluation of session-based and session-aware recommenders

As we have discussed in Chapter 2, the whole user history can be frequently seen as the concatenation of sessions ordered in time. In principle, any of the aforementioned evaluation protocols can be used with session data, by considering each session as a sequence of interactions independent from any other one. However, in this way we loose the reference of the user. For this reason, this approach is typically used in session-based recommenda-

tion, in which the activity of the user in the current sessions is independent from her past one.

In session-aware recommendation, instead, the historical activity of the user matters when generating recommendations int the current session. Therefore, the sequence-aware recommender can consider a fraction of the past user's sessions to gather additional details on her historical, long-term tastes and to adapt to sudden drifts in her interests. the evaluation procedure must be adapted to this new scenario.

We briefly describe the framework for the evaluation of session-aware recommendation proposed in [66].

- *Dataset partitioning.* The history of every user is partitioned into training and test sequence datasets $\mathcal{S}_{\text{TRAIN}}$ and $\mathcal{S}_{\text{TEST}}$ as usual. Splits within-sessions are not allowed. Sessions are partitioned at user-level and temporally. Alternatively, the last-k sessions of each user are used for testing and the remaining ones for training.

- *Evaluation protocol.* The evaluation is performed as session-level. Any of the evaluation protocols used for the evaluation of a generic sequence-aware recommender (sequential evaluation, sequential evaluation with look-ahead and set evaluation) can be used to evaluate the quality of recommendation within sessions. Specifically, the evaluation of a test session σ_m^u of the user u requires the definition of the user profile p^u and the ground truth q^u with either LLO or KFI [5]. Additionally, a number $c \geq 0$ of sessions before σ_m^u is added to p^u to add short-term contextual information to the information gathered from the current session ($c = 0$ in session-based recommendation).

  In summary, for every user session σ_m^u in $\mathcal{S}_{\text{TEST}}$, we first define the user profile p^u and the ground truth q^u in σ_m^u; then:

  i) Add $c \geq 0$ sessions previous to σ_m^u to p^u;
  ii) Evaluate the recommendation quality on q^u given p^u with one of standard evaluation protocols.

2.6.2 Online evaluation

Online evaluation analyzes the behavior of the recommender system in a real-life scenario by directly monitoring the activity of the users, typically by means of A/B testing. This allows to analyze the impact of the RS with much more detail. However, the analysis of large amounts of users on a real

[5] User index u added for clarity.

system is a costly procedure, and results are hardly reproducible out of the original environment used for the evaluation. Still, it is generally agreed by the community that online evaluation provides more significant insights on the real impact of a recommendation engine than what offline evaluation does. Only a few works in the literature of sequential recommendation adopted online evaluation. [123] used online evaluation to asses the quality of a sequential recommendation algorithm based on Markov Decision Processes. The online study involved 900 real users of an online book store for a period of about 2 years. The MDP was used to generate recommendations online. However, to evaluate the impact of different MDP policies, the authors run a simulation from the data collected online.

[96] perform an online evaluation of a radio-station recommender system based on MDPs by means of A/B testing over 70 users. The metrics used for the evaluation are the percentage of tracks listened and the daily usage of the system.

Finally, [53] evaluate a sequential recommender based on item embeddings over online Yahoo mail traffic in a standard A/B testing scenario, and study the impact of the proposed recommender system in term of Click Trough Rate (CTR) and Yield Rate (YD).

Part II

User Study

CHAPTER 3

Session-based Hotel Recommendation: a User Study

In this chapter, we present a user study on session-based recommendation in the hotel booking domain. The goal of this study is to provide a first intuition of the effectiveness of personalized recommendations, specifically based on a simple sequence-aware approach, in session-based scenarios. For this reason, we will not focus on the design of new algorithms for sequence-aware recommendation in this chapter (it will be the subject of the following chapters). Instead, we will analyze in detail in the perceived impact of recommendations on users in a real session-based scenario.

We consider the domain of hotel booking for the following reasons: (i) hotel booking is strongly session-based, i.e., most Online Travel Agencies (OTA) allow unregistered and anonymous users to use their services, thus past user information is rarely available; (ii) hotel booking is a so-called "bounded domain", i.e. hotels have a maximum "capacity" and item availability is variable. As we will see, this fact strongly correlates with the perceived usefulness of recommendations by users.

For to the purposes of our study, we used traditional collaborative filtering and content-based algorithms together with *implicit elicitation* to gen-

erate recommendations within the user session. Implicit elicitation is used to build a "temporary" user profile from the interactions of the user with the UI; we combined it with simple time-decay to consider the order of the actions of the user in the session and to reduce the importance of the less recent interactions on the recommended items.

Secondly, we have considered a specific condition of the hotel booking domain, that of *bounded availability*. There are domains where all items are always available, regardless of the number of users who have already "consumed" them, and others where the possibility of actual consumption depends on contextual circumstances, e.g., time. In the video-on-demand or e-book domains, for example, the electronic format of products allows for a potentially infinite number of customers to consume them at any time. In the e-tourism domain, a hotel or a tourism service can be consumed only by a limited number of users, hence the same item may become unavailable at some points of time (e.g., during high season). In "bounded" domains, where the "capacity" of products is constrained and item availability is variable, also the quality of the actual offer of items is not constant: as the number of available items decreases, the quality of the remaining ones tends to decrease, because the items that "are gone" are typically the "best" ones.

We compared the impact of non-personalized and personalized recommendations both in the scenario of unbounded and bounded availability of hotels. We evaluate the perceived quality of recommendations in terms of different subjective metrics, such as the user satisfaction and trust, and objective metrics, such as the average cost per night and the number of explored hotels.

The results of this work were published as a short paper at the *ACM Recsys 2013* conference [30], and as poster at the *WWW 2014* conference [29]. This chapter is an extended versions with additional considerations on the implicit elicitation mechanism and on the experimental results.

The rest of this chapter is organized as follows. In Section 3.1, we present the related works. In Section 3.2, we present the general method of the study. In Section 3.3, we describe the implicit evaluation procedure. In Section 3.4, we present our methodology to simulate the bounded availability condition. In Sections 3.6 and 3.7, we present the experimental results and the discussion. Finally, in Section 3.8, the conclusions of the work are drawn.

3.1 Background and related work

An implicit assumption of most existing studies on Recommender System (RS) evaluation is that all items are always available, regardless the number of users who have "consumed" (bought, used) them. Unlimited availability of items is typical of many sectors that are relevant for RSs. In the video-on-demand or e-book business, for example, the electronic format of products allows a potentially infinite number of customers to consume them at any time.

Still, in many domains items are or involve physical resources that have constrained "capacity", i.e., the same product can be consumed by a limited number of users. Examples are clothing, events, or travel services. The possibility for the customer of consuming items of this type depends on *contextual circumstances* and *varies over time*. In e-tourism, for instance, a hotel may become unavailable during high season or when the booking time is close to the desired time of usage.

We are interested to study the recommendation process and its evaluation in what we call *bounded domains* which are characterized by the constrained capacity of items and the temporal nature of item availability: whenever the capacity of an item has been consumed, the recommender system must consider that item as "*missing*".

The interest of missing items of this kind originates from the observation that they are not missing at random: as the number of available items decreases, the quality of the remaining ones tends to decrease, as the items that "are gone" are typically the "best ones". For instance, the hotels that are booked first in high season are normally the optimal ones in terms of price/comfort rate, while the remaining hotels are probably the most expensive or the worst located. Under this assumption, our goal is to investigate *if and how missing (e.g., consumed) items affect the quality of recommender systems trained on popularity-biased datasets (e.g., datasets with a small number of very popular items and a large number of much less popular items).*

A number of studies focused on the similar but yet very different "missing ratings" problem, which is related to the sparsity of user-rating matrixes and addresses the question if and when missing ratings should be considered as a negative, positive or neutral user feedback when training [88] and evaluating [107] a recommender algorithm.

Other works explore the evolution of RSs as a pure additive process: new users join the system, new content is added, new ratings are provided [73] [34]. Tracking the evolution of user preferences and encompass-

ing behavioral patterns of user rating in the computational model allows achieving significant improvements on recommendation accuracy [73].

Previous studies in unbounded domains show that high-rated items introduce biases in the design and evaluation of recommenders because of the so called popularity effect and positivity effect [107]: the majority of high ratings are condensed in the small fraction of the most popular items, or short-head.

The findings of a wide off-line study [19] pinpoint that when the user rating distribution is not taken into account, the accuracy of non-personalized algorithms is comparable to the performance of sophisticated personalized algorithms. When the most popular and widely ranked items in the short head are removed personalized algorithms are ranked higher than the non personalized algorithm.

3.2 The Study: General Method

In the following steps we compare the perceived quality of personalized and non personalized recommendations in "normal" conditions of full availability of items with the situation "*when the best are gone*". We further validate the results by analyzing the impact of a non-random consumption processes on the statistical properties of the dataset and, indirectly, on the accuracy of recommendations.

3.2.1 Instrument

For the purpose of our study, we have developed *PoliVenus*, a web-based testing framework that can be easily configured to facilitate the execution of controlled empirical studies in on-line booking services. PoliVenus implements the same layout of Venere.com [1] online portal and simulates all Venere.com functionality except payment functions. The PoliVenus framework is based on a modular architecture, can operate *with* and *without* recommendations, and can be easily customized to different datasets and 20 different recommendation algorithms.

PoliVenus offers the same user experience as the real Venere.com portal. This is our *baseline* experimental condition. Filtering the catalogue according to hotel characteristics (e.g., budget range, stars, accommodation type, city area), users are presented with a list of hotels meeting such characteristics that are ordered according to the editor's ranking criteria (e.g., best

[1] Venere.com is a subsidiary of Expedia (`https://www.expedia.com/`) has now become Hotels.com and it is accessible at `https://it.hotels.com/`.

sellers, marketing strategies). Users can explore the various hotels, check availability, and modify the desired attributes of accommodations.

Without recommendations, hotels are listed by default according to the editor's ranking, although users can change the default sorting at any time according to service level or price. With recommendations, hotels are ordered according to the recommender system ranking.

The user profile required by personalized algorithms to generate recommendations is based on the user's current interaction with the system (*implicit elicitation*). This choice is motivated by three main reasons:

- we want to support users who have no rating history or who are not interested in logging into the system (session-based recommendation);
- we are interested in exploring a smooth integration of personalized recommendations in existing online booking systems; to enable explicit elicitation would require the introduction of an intrusive add-on;
- according to a large number of works, lower effort of implicit elicitation (as compared to explicit elicitation) increases the perceived effectiveness of recommendations [42] [50] [108].

The details of the elicitation mechanism are described in Section 3.3.

3.2.2 Dataset

Venere.com has made us available with a catalog of more than 3,000 hotels and 72,000 related users' reviews. Each accommodation is provided with a set of 481 features concerning, among the others: accommodation type (e.g., residence, hotel, hostel, B&B) and service level (number of stars), location (country, region, city, and city area), booking methods, average single-room price, amenities (e.g., spa), and added values (e.g., in-room dining). User's reviews associated to each accommodation consist of numeric ratings and free-text.

We have enriched the original Venere.com dataset with additional reviews extracted from the TripAdvisor.com web site using a web crawling tool. Table 3.1 reports the detailed statistics of the dataset used in our experiments.

3.2.3 Structure of the study

The study is organized in four steps, overall involving 382 tested subjects.

	Total (Venere + Tripadvisor)	Venere	Tripadvisor (crawled)
Hotels	3,100	3,100	3,100
Users (reviewers)	210,000	72,000	138,000
Reviews and ratings	246,000	81,000	165,000
Hotel features	481	481	N/A

Table 3.1: *Dataset used in the study*

- The *first step* consists of a preliminary user experiment (240 users) devoted to identify the "most appropriate" *personalized* algorithm to be used in our final case study.
- In the *second step* the results of step 1 are investigated from a different perspective in order to design the implicit elicitation mechanism to be used in our final case study.
- This *third step* explores the process through which users select the items they choose to "consume" and defines the technique to simulate the "best are gone" condition (i.e., which hotels to set as "not available" in high season).
- Finally, in the last *fourth step* we compare the effects of a non personalized algorithm against the personalized algorithm chosen in step 1 and explore how missing items influence in the perceived quality. The comparison has been performed with an *online study* (142 users).

Step 1 is described in details in [31] and consisted of an empirical study involving 240 users. In short, we considered three algorithms one content algorithm (*DirectContent*), one collaborative algorithm (*PureSVD*), and one *interleaved hybrid algorithm* that combines the previous two algorithms. The results showed that the adoption of the *hybrid* algorithm significantly increases the perceived quality of the user experience, hence we adopted this algorithm for our following research.

3.3 Implicit Elicitation

In *step 2* the results of step 1 have been used to define the implicit elicitation mechanism. The implicit elicitation mechanism adopted in PoliVenus is the following: whenever a user interacts with an object on the interface, the system assigns a score to the hotel related to that object (e.g. link, button, map, picture, etc.). With all these *signals*, PoliVenus builds the user profile for the current user session: the user profile contains implicit hotel ratings,

where each rating is computed as a linear combination of all the signals generated for that hotel. The user profile is continuously updated with every new signal and the list of recommended hotels is updated accordingly.

In order to consider the *sequentiality* of user interactions and to give more importance to the most recent interaction, we have employed an exponential decay function to the predicted ratings. Whenever a new signal enters in the user profile, all previous ratings are divided by a damping factor. The magnitude of the damping factor controls the decay rate.

In our work, we use users' signals to estimate explicit ratings that will be later used by other recommender algorithms. Each object f is associated with a weight w_f measuring the extent to which the interaction with item i through object f expresses an interest for item i. Explicit ratings can be estimated by the rule

$$\hat{r}_{ui} = \sum_{f} w_f s_{uif} \tag{3.1}$$

where s_{uif} measures the number of user's u interactions with object f for item i.

In order to give more importance to the most recent interaction, whenever a new signal f on item i is collected for user u, all other signals for the same user are divided by a damping factor h.

$$\begin{aligned} s_{uif} &\leftarrow s_{uif} + 1 \\ s_{ujg} &\leftarrow s_{ujg}/h \quad \text{(for } j \neq i \text{ and } g \neq f\text{)} \end{aligned}$$

Parameters w_f and h have been learned by using the interactions of 240 users collected with the study described in Step 1. For each user u we assume that, by booking hotel i, the user is expressing a positive opinion for that hotel, i.e., r_{ui} = 5. In order to learn the w_f and h parameters we minimize the squared error

$$h^*, w^* = \underset{(h,w)}{\operatorname{argmin}} \left(r_{ui} - \sum_{u,i,f} w_f s_{uif} \right)^2 \tag{3.2}$$

The learned weights w^* and the decay rate h^* have been used to implement the implicit elicitation mechanism adopted in step 3.

3.4 Defining the Short-Head

During high season periods it could be difficult for a user to find a room, since most of the best hotels are already booked and the remaining ones

may be in the less attractive locations of the city or may offer low quality services. Furthermore, during high season hotels have higher prices than in low season, due to the higher request. We refer to this scenario as *the best are gone* scenario.

It is not obvious how to reproduce this scenario in order to make our simulation realistic in user experiments. Intuitively, two are the main features that distinguish low and high season: *prices variation* and *room availability variation*. The first aspect – *prices variation* – is easily addressed as *Venere.com* provided us with both low season and high season prices. In this section we describe *which* and *how many* hotels have been removed from the initial set in order to simulate the second aspect – *room availability variation*.

Experience tells us that best hotels are the first to be booked, and consequently the first to become unavailable in high season. In order to define which are "the best" hotels, we can use two different metrics. According to the common sense, the best hotels are the ones with the largest *average rating* , defined as

$$\overline{r}_{ui} = \frac{\sum_u r_{ui}}{n_i} \tag{3.3}$$

where r_{ui} is the rating from user u to item i, and n_i is the number of users who rated item i. However, average ratings computed over a larger support are considered more reliable by the users. Therefore, *popularity* can be used as an alternative metric to define the best hotels. The two metrics are not necessarily correlated, as low popularity may come along with a high hotel rating and vice versa. To overcome this ambiguity, we adopted the definition of *shrunk average rating* introduced in [11]

$$\tilde{r}_{ui} = \frac{\sum_u r_{ui}}{n_i + k} \tag{3.4}$$

where k is a shrink constant that controls the support of the estimate. For $k = 0$ hotels are ranked according to the traditional definition of average rating. For $k \to \infty$ hotels are ranked according to their popularity. In our experiments we set $k = 10$.

Hotels are sorted based on their shrunk ratings and the topmost hotels that capture 66% of ratings are considered the "best". In our dataset there are 1500 "best" hotels, out of 3100 hotels overall (almost 50%). This number is close to the percentage of fully booked hotel as reported by *Venere.com* during high season periods in Rome. The "best" hotels are considered fully booked in the high-season scenario and are removed from the dataset during the on-line recommendation.

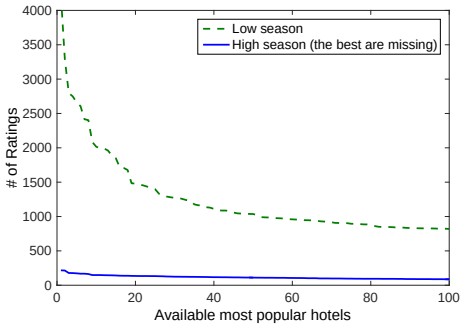


Figure 3.1: *Distribution of the number of ratings for the first 100 most popular hotels.*

Figure 3.1 reports the popularity of the 100 most popular hotels, for both the low and high season scenarios. Hotels (horizontal axis) are sorted in descending order of popularity. The dashed line represents the number of ratings in the original, unmodified dataset (i.e., the low season scenario). The solid line represents the most popular hotels which remain in the dataset when the best hotels have been removed (i.e., the high season scenario). It is clear from the figure that the high season scenario removes from the dataset the short-head of the distribution.

3.5 The Study

3.5.1 Dependent Variables

We model the user centric *effects* on quality of recommendations that can be associated to the introduction of personalized recommendations using two types of constructs: i) *subjective variables* - attributes resulting from the user's perception and judgment of the decision making activity; ii) *objective variables* objectively measurable attributes of the decision process and outcome. The complete list of variables is presented in Table 3.2. Subjective variables are measured using a web based questionnaire, proposed to participants at the end of their reservation process. Objective variables are assessed using interaction log data.

3.5.2 Independent Variables

The effects of recommendations are explored under different experimental conditions, defined by the combination of two manipulated variables*:* *hotels availability* and *recommendation algorithm*.

Variable	Type	Description	Question
Choice satisfaction	Subj.	The subjective evaluation of the reserved hotel in terms of quality/value for the user	Are you satisfied with your final choice?
Trust	Subj.	The perceived degree of matching between the characteristics of the chosen hotel emerging from the use of the system and the real characteristic of the accommodation	Will the description of the chosen hotel match its real characteristic?
Hotel price	Obj.	The cost of one night for the reserved hotel	N/A
Extent of hotel search	Obj.	The number of hotels that have been searched, for which detailed information has been acquired	N/A

Table 3.2: *Dependent variables used in our studies*

For hotels availability we consider two possible values: *low season* (or full availability) and *high season* ("when the best are gone").

Our study considers three recommendation techniques: *Editorial*, *Hybrid* and *Popular*.

- *Editorial*. We consider as *baseline* "algorithm" the most common approach of online booking systems, which ranks hotels based on some marketing strategy. We adopted the default ranking of Venere.com, which is (mainly) based on the number of users who booked the hotel.

- *Hybrid*. This technique generates a list of recommended hotels interleaving the results from *PureSVD* and *DirectContent* algorithms. Interleaving has been proposed in [19] with the name "mixed hybridization" and, although trivial in its formulation, has been shown to improve diversity of recommendations. *PureSVD* is a collaborative algorithm based on matrix-factorization; previous research shows that its accuracy is one of the best in the movie domain [31]. *DirectContent* recommends hotels whose content is similar to the content of hotels the user has rated [82]. Content analysis takes into account 481 features (e.g., category, price-range, facilities), the free text of the hotel description, and the free text of the hotel reviews. *DirectContent* is a simplified version of the LSA algorithm described in [11].

- *Popular.* This technique generates a ranked list of items based on the shrunk average rating .

3.5.3 Study execution

Our main research audience is represented by users aged between 20 and 40 who have some familiarity with the use of the web and had never used Venere.com before the study (to control for the potentially confounding factor of biases or misconceptions derived from previous uses of the system). The total number of recruited subjects who completed the task and filled the questionnaire by the deadline was *142*. They were equally distributed in the 6 experimental conditions. We recruited participants from current students, ex-alumni and administrative personnel at the School of Engineering and the School of Industrial Design of our university. They were contacted by e-mail, using university mailing lists. The invitation included the description of the task to be performed and the reward for taking part in the study. Users were not aware of the true goal of the experiment.

To encourage participation, and to induce participants to play for real and to take decisions as they would actually do when planning a vacation, we used a lottery incentive [106]. Participants had the chance of winning a *prize*, offered to a randomly selected person who completed the assigned task and filled the final questionnaire by a given deadline. The prize consisted of a coupon of the value of 100€to be used to stay in the hotel fictitiously reserved using PoliVenus.

All participants were given the following instructions: "*Imagine that you are planning a vacation in Rome and are looking for an accommodation during Easter season; choose a hotel and make a reservation for at most two 2 nights; dates and accommodation characteristics (stars, room type, services, and location) are at your discretion. After confirming the reservation (simulated), please complete the final questionnaire*".

After accessing PoliVenus and agreeing on the study conditions (lottery participation and privacy rules), each participant was automatically redirected to the homepage of the PoliVenus reservation system and randomly assigned to one of the six experimental conditions. After committing the reservation, the user was directed to the questionnaire page containing 10 questions, a subset of which is reported in Table 3.2.

3.6 Results

In this section we analyze and discuss the results of the final study where we compare the situation "with all items" against the one when "the best

are gone".

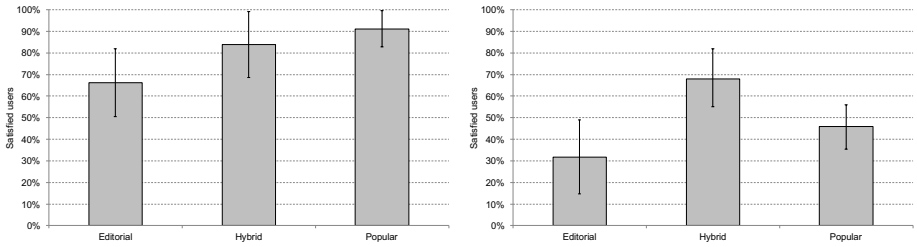
We first polished the collected data by removing the ones referring to subjects who showed apparent evidences of gaming with the testing system (e.g., those who interacted with the system for less than 2 minutes) or left too many questions unanswered. In the end, we considered the data referring to *125* participants. They were almost equally distributed in the six experimental conditions, each one involving a number of subjects between *20* and *24*.

Analysis of variance (Anova) suggests that the six different experimental conditions (type of algorithm and hotel availability) have a significant impact ($p<0.05$) on all the variables. We ran multiple pair-wise comparison post-hoc tests using Tukey's method on the mean value of the dependent variables. The results are shown in Figures 3-7, together with 95% confidence intervals.

Figure 3.2 describes *user satisfaction* in the 6 experimental conditions for the low season and high season scenarios. In the low season scenario (Figure 3.2a), more than 90% of the users are happy with their choice (percentage of "yes" answers to the question about choice satisfaction) with top popular non-personalized recommendations; and the satisfaction measure is significantly better compared to the baseline Editorial presentation of hotels.

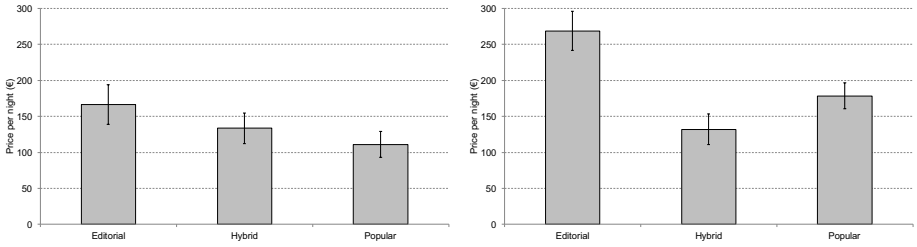
In the high season scenario users are overall less satisfied than in the low season in all 3 experimental sub-conditions. This is not surprising as users may interpret the scarcity of resources in a given period as a weakness of the catalogue of services and ascribe the phenomenon to the service provider rather than an objective contingent situation. For users receiving editorial and top popular recommendations the percentage of satisfied users reduces to half its value: satisfied users drop from 60% down to 30% in the editorial case, and from 90% down to 45% in the top popular case. In contrast, *users receiving personalized recommendations in the high season situation are now the most satisfied (70%) and their number does not significantly differ from the low season scenario.*

It is interesting to compare these results against an objective variable, the average price per night of the hotels reserved by the users. In non-personalized conditions there is a statistically significant negative correlation between hotel price and satisfaction. In the low season scenario (Figure 3.3a) users who receive popular recommendations and are more satisfied spend significantly less, on average, than users in the baseline scenario: 100€vs. 150€per night. Again, no statistically significant difference emerges in conditions of personalized recommendations (where the cost of



(a) *Low season* **(b)** *High season*

Figure 3.2: *Percentage of satisfied users.*



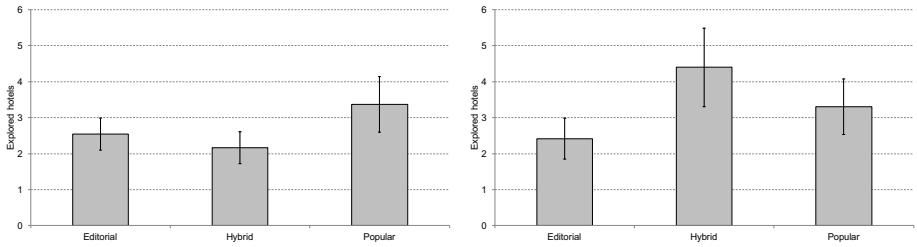
(a) *Low season* **(b)** *High season*

Figure 3.3: *Average cost per night.*

booked hotels is between 100€and 150€). In high season, when most of the hotels are fully booked and the average cost per room increases, the average price of booked hotels increases by more than 70% in the editorial condition and by approximately 50% in presence of top popular recommendations (Figure 3.3b). Still, it is not significantly different from the low season scenario when users' decision is supported by personalized recommendations.

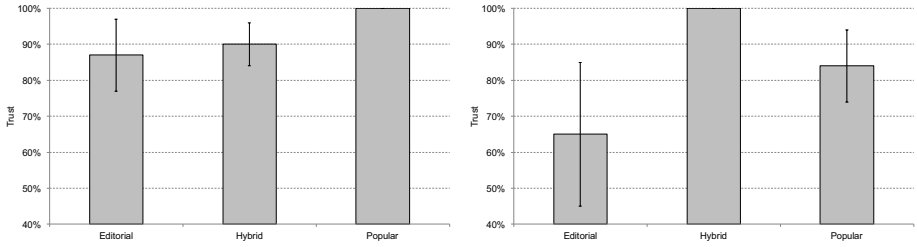
Concerning effort, Figure 5 plots the *task execution time* in the different experimental conditions. This finding confirms the intuition that searching for hotels in the low season period intuitively takes shorter time than in the high season period when most of the best hotels are fully booked and the decision making process is more complex.

Less intuitively, in both low and high season conditions, the most satisfied users invested more time on the decision process than less satisfied users. The findings on the effort measured with a different variable *average number of hotels "explored"* by the users – are consistent with the results on task time. In the low season scenario, Figure 3.4a shows that users who receive top popular recommendations and are the most satisfied, explore the largest number of hotels. The same happens in the high season scenario for



(a) *Low season* **(b)** *High season*

Figure 3.4: *Number of explored hotels.*



(a) *Low season* **(b)** *High season*

Figure 3.5: *Percentage of users trusting the booked hotel*

users who receive personalized recommendations and are the most satisfied (Figure 3.4b).

Figure 3.5 highlights a correlation between trust and satisfaction. According to Figure 3.5a, the majority of the users in the low season scenario trust the booked hotel (i.e., they believe the hotel characteristics matches its description). Noticeably, all low season users receiving top popular recommendations and all high season users receiving personalized recommendations trust the system. For users receiving editorial recommendations trust drops from 90% (low season) down to 65% (high season).

3.7 Discussion

3.7.1 User-centric quality

The low season condition is comparable to the situation of potentially unlimited capacity, when all items are always available, which characterizes most domains considered by RS research. Hence it is not surprising that our results on satisfaction in the low season scenario are in line with prior findings [31] that consider the movie domain and pinpoint that the per-

ceived quality of non personalized algorithm is comparable to the one of personalized algorithms. There are a number of possible interpretations of this phenomenon, consistent with the results of our off-line study. It is well known that the opinion of the crowd has a strong persuasion effect, often times higher than individual unexpressed preferences.

When there is a large amount of products potentially satisfying the characteristics explicitly specified by the user (e.g., stars, services, location), the most important attributes of items that drive the user decision process are number of ratings and their value. Trivially, most users are biased by success, hence algorithms that are biased by popularity are better. These algorithms implement a first order persuasion criteria – product success. When the offer of products abounds, less biased algorithms that rely more on a second order persuasion criterion – personalization – are not appreciated and are less trusted: their suggestions do not match the opinion of the crowd and this mismatch can be noticed by the users just by looking at the popularity and ratings of the suggested items. It is worth noticing that the mismatch in perceived quality between personalized and non personalized recommendations that emerges from our study is stronger that in the findings of [31]. This effect may be due to the intrinsically different characteristics of the hotel booking domain with respect to the movie domain which is the subject of the studies reported in [31]. Personalized algorithms attempt to support novelty and serendipity. But, differently from other domains (e.g., movies) these attributes are not necessary important in hotel booking. Users who already stayed in a city might wish to book the same hotel used in the past if it offered a satisfactory experience. If this accommodation is available, recommendations of something new or unexpected, regardless its degree of match with personal tastes, might not be taken much into account. In contrast, users would not buy a movie twice, appreciating the recommendation of items that are novel and serendipitous.

Let us now consider what happens when short head items are missing (high season scenario). Our findings show that, in the relative ranking of algorithms, personalized recommendations are now more effective than popularity based recommendations. In conditions of scarcity of offer most popular choices are not available (i.e., short head items are missing). Most of the available items are in the longer tail, with no or few user ratings. They all appear to be "below threshold", indistinguishable from one another, with respect to the attributes of average rating and popularity that act as persuasion factors when the offer is abundant. Users are less biased by popularity and popularity-based algorithms reduce their effectiveness. At this point, other qualities of items become important, such as an acceptable match

between item characteristics and personal needs. Hence personalized algorithms which are unbiased by popularity increase their persuasion strength as they provide better support to users in discovering alternative and novel but yet satisfactory solutions.

Beside satisfaction, other results emerging from our user study are consistent with the above arguments. In the high season, "trust" and "number of explored hotels" increase and reach the top level with personalized recommendations, while they decrease with popularity-based recommendations. When the best and most obvious solutions are gone, users are forced to spend more effort in searching for items and exploring information related to the choice process (as shown also by the values of the time variable). But this burden is mitigated by the benefits of a more satisfying and trustworthy decision process. Finally, it is interesting to consider the results of the "price of booked hotels" in the different experimental conditions. It is intuitive that higher levels of choice satisfaction are related to lower price of the chosen hotel. What is more surprising, but is consistent with the above analysis and the findings on effort, is that the average cost of reserved hotels in the condition of personalized recommendation is not affected by the scarcity of offer, remaining stable in low and high season. By effect of personalized recommendations, users tend to explore more items, become more conscious of alternative offers, and seem to be more able to discover hotels at reasonable prices.

3.7.2 Validity of the study

The internal validity of our user study is supported by the accuracy of our research design and by the quality of study execution. We have carefully implemented a mechanism to randomly assign participants to the different experimental conditions. We have adopted a lottery incentive to improve the accuracy of the task's execution and offer a shared motivation to all participants. Still, the individuals' intrinsic characteristics and actual behavior always bring to an experiment a myriad of factors that can be hardly controlled. In terms of external validity, the results of our study are obviously limited to the participants and conditions used in our study. We cannot generalize our findings to contexts where other design decisions are adopted in relationship to algorithms, elicitation techniques, and interface. Still, within the scope of our specific design choices, the applicability of our results might not be confined to the specific online booking system used. Most services available in the market provide a user experience very similar to Venere.com, in terms of filtering criteria and information/navigation

structures, and it is likely that replications of our study on other systems may lead to results consistent with our findings. Finally, the high overall number of testers (240 in the first user study, and 142 in the second one) and the relatively high number of subjects involved in each experimental condition may allow us to generalize our results to a wider population of users.

3.8 Conclusions

In this chapter we focus on session-based recommendation in business sectors that we call *bounded* domains. We present a relatively simple yet effective way of building a profile of the user from the sequence of interactions in the current session based on implicit elicitation combined with time decay. With such mechanism we were able to provide personalized recommendations to new users solely on the basis on their activity in the current session.

We bring a number of contributions to RS research and practice. From a research perspective, our work considers attributes of items – constrained capacity and availability – that have received so far little attention in the RS community. Our main result is that in our case study the *subtractive effect* resulting by item consumption strongly weakens the performance of non personalized popularity based recommendations. In contrast, personalized recommendations do not exhibit such a negative behavior: *"when the best are gone"* on-line measured satisfaction and effort remain stable, while trust improves. We are not aware of other *online* experiments that consider the effects of subtractive operations on recommender quality.

Our results emphasize the importance of personalization strategies in session-based recommendation. In this chapter we focused on the implicit signals generated by the user from her interactions with the UI. However, we left out other important signals that can be used to better identify the intent of the user in the current session, such as the information hidden in the sequence of items explored by the user (we loosely addressed this aspect by introducing a time decay factor in the user profile), or the sequential relations between the content of the explored items themselves, as well as the influence of past user sessions on the current one. All these aspects cannot be easily encoded into a single, relatively static user profile vector. Furthermore, traditional collaborative or content-based algorithms, as we have seen in Chapter 2, do not provide the sufficient flexibility to adapt to the short-term changes in the user interests. Therefore, it urges the definition of new, powerful, sequence-aware algorithms for session-based

and session-aware recommendation. We will present novel solutions that address these and other research needs in the following chapters.

Part III

Novel algorithms

CHAPTER 4

Feature-rich session-based recommendation with Recurrent Neural Networks

In traditional recommender systems, it is often assumed that the complete user history is available. As shown in the previous chapter, this assumption does not hold in many real-world recommendation use cases: (1) many e-commerce sites do not require user authentication even for purchase; (2) in video streaming services users rarely log in; (3) many sites have a small percentage of returning users; (4) the user intent can change between different sessions, typical for example in classified sites. These are typical cases of session-based recommendation in which is fundamental to extract as much information as possible from the interactions of the user in the session given the absence of user profiles.

In this chapter, we consider the problem of next-item recommendation in session-based scenarios with click-stream data. The interactions in every session are represented by the stream of temporally ordered item-IDs clicked by the user. The goal is to predict the right item the user will click next, given her past activity in the current session.

Beside item-IDs, we also take into account the characteristics of the

items that have been clicked. In real systems items usually come with rich feature representations. In streaming services, videos are usually associated to a title and a small thumbnail that are explicitly chosen to capture users' attention. In classified advertisement, announcers usually provide a short title and a description of the product being sold, aiming to attract potential buyer's interest while she is browsing among many other – possibly similar – announcements. Features become particularly important in a session modeling setting where historical user specific data is missing or has no importance. Given the absence of user profiles, in session-based recommendation it is vital to draw as much information as possible from the interactions of the user in the session, and such features should be utilized to aid the session modeling process.

We utilize Deep Learning techniques both to extract high quality features from visual information and to model the sessions. We model individual user sessions as sequences of clicks with Recurrent Neural Networks (RNNs) and extract visual features from video thumbnails with pretrained Convolutional Neural Networks (CNNs). We also extract features from text via bag-of-words.

The jointly modeling of sequences of clicked item-IDs and item features is addressed by introducing novel parallel RNN (p-RNN) architectures. We discuss about the optimal training procedures for session modeling with p-RNNs. We run an extensive evaluation of the model in two real-life scenarios – video and product recommendation – over two large datasets of two commercial websites provided by Gravity R&D [1]. We compare against item-KNN, which is today's industrial standard for session-based recommendation.

Part of the results of this work were presented as a full paper at the *ACM RecSys 2016* conference [62]. This chapter is an extended version of this paper, with additional details on the training procedure, the hyperparameter tuning and experimental results that were not covered in the paper.

The rest of this chapter is structured as follows. In Section 4.1 we present the necessary background on Recurrent Neural Networks, on session-based recommendation with RNNs and on feature-based recommendation. In Section 4.2 we describe the feature extraction from image and text. In Section 4.3 we present the proposed p-RNN architectures and the training procedures. In Section 4.4 we present the experimental setup, the parameter tuning and the results of our experiments. Finally, conclusions are drawn in Section 4.5.

[1] http://www.yusp.com/

4.1 Background and related works

In this section, we present a brief introduction to Recurrent Neural Networks. We then describe a prior work on session-based recommendation that served as foundation of ours. Finally, we briefly describe the state-of-the-art on feature-rich recommendation.

4.1.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are neural models explicitly devised to process sequences of arbitrary length. They have been recently used with success in a variety of fields like speech signal modeling and polyphonic music modeling [28]. They differ from feedforward neural networks by the inclusion of connections that span adjacent time steps, introducing the notion of time in the model [80]. They are characterized by a distributed hidden state that is used to store information about the past efficiently, and non-linear dynamics to compute complex updates of the hidden states.

At each time step t, the RNN receives the input data point $\mathbf{x_t}$ and the previous hidden state $\mathbf{h_{t-1}}$. In standard "vanilla" RNNs, the hidden state, or activation, $\mathbf{h_t}$ is updated according to the following equation

$$\mathbf{h_t} = g\left(W_{hx}\mathbf{x_t} + W_{hh}\mathbf{h_{t-1}} + b_h\right) \tag{4.1}$$

where W_{hx} and W_{hh} are the *input* and *hidden* transformation matrices, b_h is the *hidden bias*, and g is a smooth and bounded *activation function* such as a logistic sigmoid function or hyperbolic tangent.

In short, at each time step t, the hidden state of the network $\mathbf{h_t}$ is computed as a simple weighted sum of the current input $\mathbf{x_t}$ and the previous hidden state $\mathbf{h_{t-1}}$ with a non-linear function g on top. Given the current state $\mathbf{h_t}$, the RNN outputs a probability distribution $\hat{\mathbf{y}}_\mathbf{t}$ over the next element of the sequence like in the following equation

$$\hat{\mathbf{y}}_\mathbf{t} = \mathrm{softmax}\left(W_{yh}\mathbf{h_t} + b_y\right) \tag{4.2}$$

where $\mathrm{softmax}\left(z_k|\mathbf{z}\right) = \frac{\exp(z_k)}{\sum_j \exp(z_j)}$ is the softmax function.

The network can be trained to minimize the classification error over the next element in the sequence using, for example, the cross-entropy loss

$$L\left(\mathbf{y_t}, \hat{\mathbf{y}}_\mathbf{t}\right) = -\sum_t \sum_k y_{tk} \cdot \log\left(\hat{y}_{tk}\right) \tag{4.3}$$

Activations can be stacked to increase the depth of the RNN, and complex RNNs structures can be constructed to address the characterstics of the task

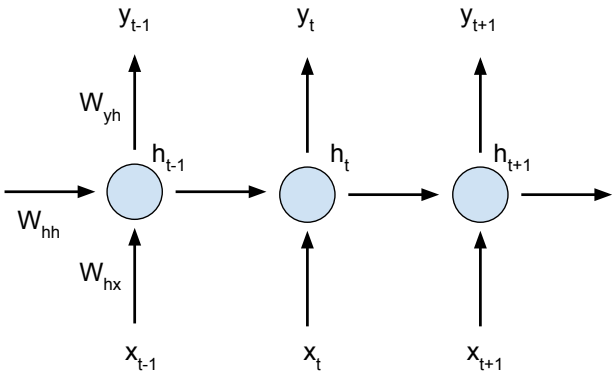


Figure 4.1: *A visualization of a Recurrent Neural Network. The weights of the network W_{hx}, W_{hh} and W_{yh} are the same at every time step.*

that is modeled (e.g. text classification, sequence to sequence translation, text generation). See [80] for a comprehensive review on sequence learning with RNNs.

When all the components of the RNN and the loss function are differentiable, the network can be trained in a gradient-based fashion through the back-propagation [115]. However, learning with RNNs is known to be very challenging due to the problems of *vanishing* and *exploding* gradients that occur with backpropagation across many steps [100]. These issues can be addressed by means of advanced learning algorithms such Truncated Back-Propagation Through Time (TBPPT) [140], gradient clipping and second-order methods (e.g. [36, 44]).

Additionally, "modern" RNNs use complex hidden dynamics to mitigate the aforementaioned issues. We briefly describe here the two most used variants, namely Long Short-Term Memory (LSTM) [64] and Gated Recurrent Unit (GRU) [27]. Both models use complex dynamics to learn when and how much to update their hidden state, which in turn allows to mitigate the impact of vanishing gradients. They both replace hidden units with *memory cells*. LSTMs and GRUs basically differ only in the complexity of their cells.

Long Short-Term Memory (LSTM)

LSTMs have been subject to many modifications since its introduction in [64]. We present here the version of [52]. The LSTM (Figure 4.2a)

maintains a memory cell $\mathbf{c_t}$ at time t. The activation $\mathbf{h_t}$ is computed as

$$\mathbf{h_t} = \mathbf{o_t} \tanh(\mathbf{c_t}) \tag{4.4}$$

where $\mathbf{o_t}$ is the *output gate* that modulates how much information in the memory cell is exposed. The output gate is computed as

$$\mathbf{o_t} = \sigma(W_{ox}\mathbf{x_t} + W_{oh}\mathbf{h_{t-1}} + W_{oc}\mathbf{c_t}) \tag{4.5}$$

where $\sigma(\mathbf{z}) = \frac{1}{1+\exp(-\mathbf{z})}$ is the sigmoid function. The content of the memory cell is updated by partially forgetting the previous memory and adding the new memory:

$$\begin{aligned}\tilde{\mathbf{c}}_\mathbf{t} &= \tanh(W_{cx}\mathbf{x_t} + W_{ch}\mathbf{h_{t-1}}) \\ \mathbf{c_t} &= \mathbf{f_t}\mathbf{c_{t-1}} + \mathbf{i_t}\tilde{\mathbf{c}}_\mathbf{t}\end{aligned}$$

where $\mathbf{f_t}$ and $\mathbf{c_t}$ are the *forget gate* and *input gate* respectively, that are computed by

$$\begin{aligned}\mathbf{f_t} &= \sigma(W_{fx}\mathbf{x_t} + W_{fh}\mathbf{h_{t-1}} + W_{fc}\mathbf{c_{t-1}}) \\ \mathbf{i_t} &= \sigma(W_{ix}\mathbf{x_t} + W_{ih}\mathbf{h_{t-1}} + W_{ic}\mathbf{c_{t-1}})\end{aligned}$$

Gated Recurrent Unit (GRU)

GRUs (Figure 4.2b) are simpler versions of LSTMs. The activation of the GRU is a linear interpolation between the previous activation and the candidate activation $\tilde{\mathbf{h}}_\mathbf{t}$:

$$\mathbf{h_t} = (1 - \mathbf{z_t})\mathbf{h_{t-1}} + \mathbf{z_t}\tilde{\mathbf{h}}_\mathbf{t} \tag{4.6}$$

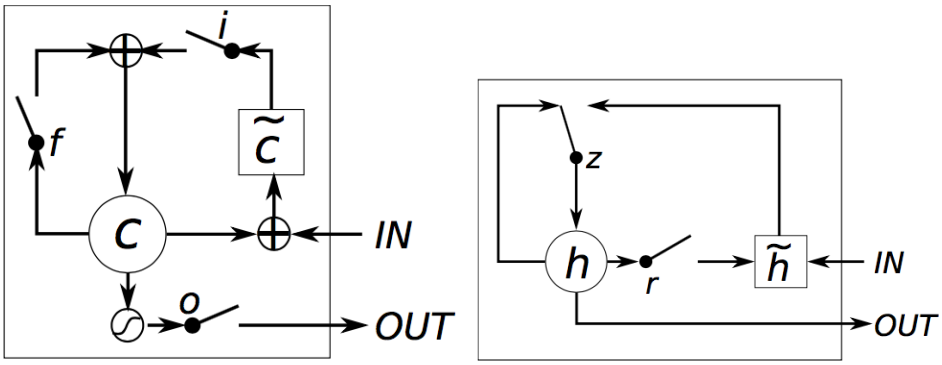
where the *update gate* $\mathbf{z_t}$ and the candidate activation $\tilde{\mathbf{h}}_\mathbf{t}$ are given by:

$$\begin{aligned}\mathbf{z_t} &= \sigma(W_{zx}\mathbf{x_t} + W_{hz}\mathbf{h_{t-1}}) \\ \tilde{\mathbf{h}}_\mathbf{t} &= \tanh(W_{hx}\mathbf{x_t} + W_{hh}(\mathbf{r_t} \odot \mathbf{h_{t-1}}))\end{aligned}$$

and finally the *reset gate* $\mathbf{r_t}$ is given by:

$$\mathbf{r_t} = \sigma(W_{rx}\mathbf{x_t} + W_{rh}\mathbf{h_{t-1}}) \tag{4.7}$$

The simpler cell dynamics of GRUs makes them less computationally expensive than LSTMs. Still, they have been shown to maintain the same properties of LSTMs in many sequence learning tasks [27]. For these reasons, GRU is the model of choice in this work. In order to simplify the understanding of the following sections, we will refer to GRU simply as RNN, unless explicitly noted.



(a) *Long Short-Term Memory* **(b)** *Gated Recurrent Unit*

Figure 4.2: *Illustration of (a) LSTM and (b) GRU cells from [27].*

4.1.2 RNNs for session-based recommendation

The first successful application of RNNs to session-based recommendation to our knowledge in presented in [61]. The input to the network is the current item of the session while the output is the next item in the session. Inputs are one-hot encoded, i.e., the input vector has length equal to the number of items in the catalog and only the entry corresponding to the current item is one while others are set to zero. An additional embedding layer can be used in input, but empirically it does not show any performance benefit.

The architecture of the network (Figure 4.3a) is composed of GRU layer(s) and additional feedforward layers that can be added between the last layer and the output. The output of the network is the predicted preference on the next item, i.e., the likelihood of being the next item in the sequence. In addition, GRU layers can be stacked and the input can be propagated to layers deeper in the network.

To address the peculiarities of the recommendation domain, the authors introduce three modifications to the training procedure, namely *session-parallel mini-batches*, *output sampling* and *ranking loss*.

- **Session-parallel mini-batches.** RNNs in NLP are generally trained with in-sequence mini-batches, e.g., by segmenting sequences into fragments with a sliding window over the words in the text and concatenating such fragments to generate a mini-batch. However, user sessions can be of very different length, and breaking sessions down into fragment would not allow to capture how the session evolves over

time. For these reasons, the authors propose session-parallel mini-batches (Figure 4.3b). Sessions are first ordered, then the first and second events of the first B sessions form respectively the input and the output mini-batches, and so on. Forthcoming sessions are added to the mini-batch as old sessions end and the corresponding hidden states in the RNN reset.

- **Output sampling.** The computation of the output activation of the network (Equation 4.2) scales linearly with the number of items. This can be extremely costly when dealing with large catalogs of tens of thousands or even millions of items. Therefore, the authors propose to sample the output and to compute the score only for a small subset of the items. The sampling is performed efficiently by considering, for every item in the mini-batch, all the other examples in the mini-batch as negative examples. This procedure considerably reduces the complexity in training and has the desirable property of being a popularity-based sampling approach [110], since the likelihood of an item being in the other training examples is proportional to its popularity.

- **Ranking loss.** Next event prediction can be interpreted as a traditional classification task. However, in recommender systems ranking losses generally outperform other approaches. For this reason, the authors propose to use pointwise and pairwise ranking losses. Pointwise losses, such as cross-entropy, estimate the ranking of items independently and learn how to push the relevant item up in the recommendation list. Pairwise losses, such as BPR [111], instead consider the score and ranks of pairs of relevant (positive) and non-relevant (negative) items and optimize for the relative ordering between pairs.

  In the training procedure, the loss is averaged over samples of N_S items ($N_S = B$ in the case of training with parallel mini-batches). $r_{s,i}$ and $\hat{r}_{s,i}$ are the actual and predicted relevance score for the next-item i in the session s. Notice that, in the case of binary implicit feedback like in click-stream data, $r_{s,i} = 1$ iff i is the item clicked by the user in the next event in s and 0 otherwise. The pointwise and pairwise losses used to train RNNs with click-stream data are:

  1. **Pointwise:**
     - *Cross-entropy.* This is the mini-batch version of the cross-entropy loss defined in Equation 4.3. The loss at a given point in the session s is computed as:

$$L_S = -r_{s,i} \cdot \log\left(\hat{r}_{s,i}\right) \tag{4.8}$$

This loss maximizes the score of the positive items without considering the negative ones. This loss can be subject to numerical instability issues due to values $\hat{r}_{s,i}$ close to 0 during training (with consequent loss values that approach infinity). This issue is commonly addressed by by adding a very small constant ϵ to L_S, e.g. 10^{-24}, which bounds the values L_S can take.

2. **Pairwise:**
   - *BPR*. This is the well-known Bayesian Pairwise Ranking [111] loss, widely used in recommendation systems. It compares the relevance score of a positive item and a sampled negative item. The loss at a given point in the session s is defined as:

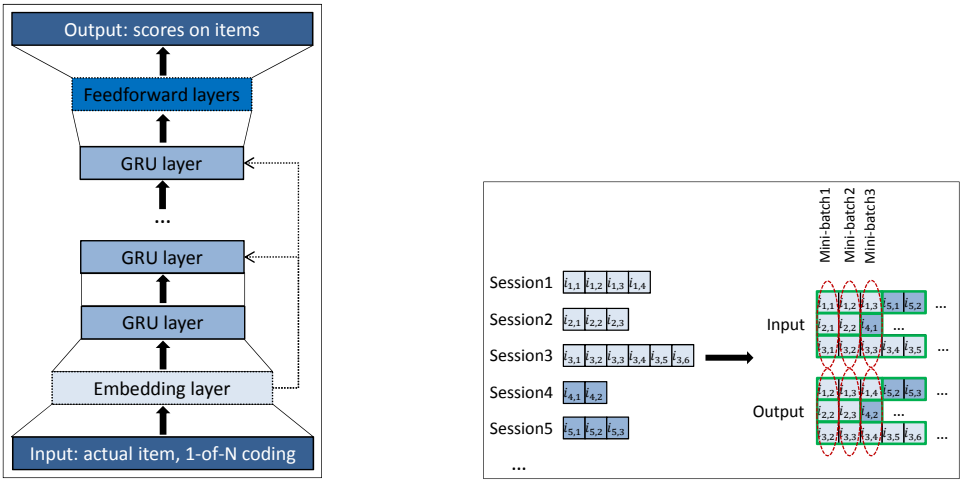
$$L_S = -\frac{1}{N_S} \cdot \sum_{j=1}^{N_S} \log\left(\sigma\left(\hat{r}_{s,i} - \hat{r}_{s,j}\right)\right) \tag{4.9}$$

     where j are the negative samples.
   - *TOP1*. This pairwise ranking loss was explicitly devised for the task. It is a regularized approximation of the relative rank of the positive item and it is computed as:

$$L_S = \frac{1}{N_S} \cdot \sum_{j=1}^{N_S} \sigma\left(\hat{r}_{s,j} - \hat{r}_{s,i}\right) + \sigma\left(\hat{r}_{s,j}^2\right) \tag{4.10}$$

     The first term in the summation is a continuous approximation of the relative rank of the relevant item $I\left\{\hat{r}_{s,j} > \hat{r}_{s,i}\right\}$ where the indicator function is replaced with a sigmoid. This induces the model to assign high scores to $\hat{r}_{s,i}$. The second term in the summation acts as a regularizer and pushes the score of non-relevant items towards zero.

The network is trained with Adagrad [44] with momentum [130] for a fixed number of epochs (10). The final results of the paper show that single-layer RNNs trained, parallel mini-batches and TOP1 loss significantly outperform the Item-KNN baseline and other non-personalized and sequence-agnostic baselines such as Popularity by large margins (+20-24% in Recall and +15-20% in MRR over Item-KNN) on two large datasets, hence establishing a new state-of-the-art in session-based next-event recommendation.



(a) *Architecture* **(b)** *Parallel mini-batches*

Figure 4.3: *Architecture and parallel mini-batches.*

4.1.3 Feature-rich Recommender Systems

The integration of item features into recommendation systems sets its roots into Content-based recommendation [113]. Items can be associated with very diverse types of features, like human-curated metadata [125] and tags [83]. While item features are important resources in item cold-start scenarios [120], it is acknowledged by the community that collaborative approaches quickly outperform content-based ones as soon as user interactions become sufficiently dense [103]. This fact is primarily imputable to the higher quality of the user and item descriptors that can be extracted from interaction data. Additionally, metadata and tags are just abstract, structured item representations that do not really reflects what the user is actually seeing or listening, i.e., the actual content of the item.

To address the inherent mismatch between traditional item representations based on tags and metadata and the actual content of items, recently a new class of recommenders, said *feature-rich recommender systems*, has been developed. These recommenders leverage features that are automatically extracted from the unstructured item content itself and combine them with traditional collaborative and content recommendation.

In the context of video recommendation, in [37] we considered stylistic and low-level features that automatically extracted from videos in content-based recommendation. Low-level features (lighting, color and motion) capture the inherent stylistic aspects of movies that cannot be straightfor-

wardly encoded by metadata, showing significant improvements in recommendation wrt. video recommendation based on metadata.

In place of handcrafted feature descriptors, deep models can be used to extract features from unstructured content such as music or images [40, 124]. For example, convolutional deep networks have been used to extract features from music tracks that are then used in a factor model [136]. More recently [138] introduced a more generic approach whereby a deep network is used to extract generic content-features from items, these features are then incorporated in a standard CF model to enhance the recommendation performance. This approach seems to be particularly useful in settings where there is not sufficient user–item interaction information. Image features that have been extracted using convolutional networks have been used in classical matrix factorization-type CF in [58, 89] to enhance the quality of recommendations.

4.2 Feature extraction

4.2.1 Feature extraction from images with Deep CNNs

Recently, deep learning research on convolutional neural networks (CNNs) achieved breakthroughs in a variety of different image-related tasks, like object recognition, image segmentation, video classification, etc. [70, 74] even surpassing human performance on the task of object recognition [56]. Unlike other approaches, CNNs don't require prior feature extraction, since they are capable of working on the raw image data. CNNs trained on millions of images produce image features that can then be used as input in other algorithms e.g. clustering [40, 124]. These models generalize well and also perform well on images that the CNN has never encountered during training and can thus be used as generic feature extractors. This makes CNNs ideal for extracting high quality image features.

We used the GoogLeNet [131] implementation of the Caffe deep learning framework [70] to extract features from the thumbnails of the videos. The network was pre-trained as an image classifier on the ImageNet ILSVRC 2014 dataset [116] that contains 1.2M images organized into 1000 categories. The video thumbnails first had to be scaled down and cropped in order to fit the input of the network. Features were extracted from the last average pooling layer, extracted as the value of the *pool5_7x7_s1* layer (Figure 4.4). The feature vectors were normalized to have unitary l^2-norm. The image feature representation we end up with is a real-valued vector of length 1024.

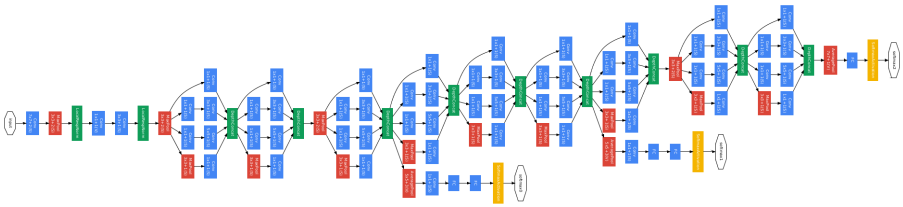


Figure 4.4: *Diagram of GoogLeNet [131]. We extracted features from the last average pooling layer pool5_7x7_s1.*

Figure 4.5 demonstrates the feature quality by showing the 3 most similar images to two query images, where similarity is defined as the cosine similarity between the image feature vectors. Given the good quality, we do not plug the CNN directly into the RNN, as it would introduce unnecessary complexity to the training and is also unpractical, because (a) this network would converge much slower as it needs to learn the model on incomplete/changing item representations; (b) the network would not be suitable for datasets with lower number of items, as 10,000s of items are not enough to leverage the full potential of the CNN; (c) retraining would take much longer. Another possibility is to use the pretrained network and fine tune the item representations during the training of the RNN. This did not make much difference in our experiments, therefore we did not use fine tuning.

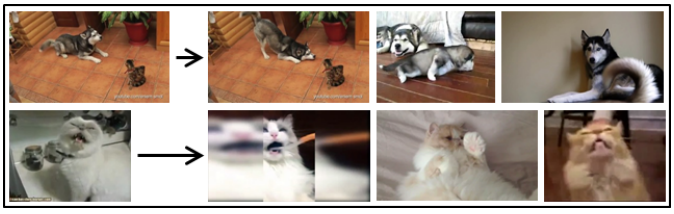


Figure 4.5: *Top3 similar images to query images, based on cosine similarity of image feature vectors.*

4.2.2 Feature extraction from unstructured text

Given the strict limitation on the length of the descriptions imposed by online classified advertisement platforms, advertisers usually provide rather concise text for their items. The main goal of the description is to attract the attention of potentially interested users. Therefore, descriptions often

contain only the main characteristics of the item and use syntactically incorrect sentences. Moreover, it is not rare to have descriptions written in multiple languages to capture a broader audience. The majority of descriptions of our dataset used a subset of 3 main languages with a handful of less frequent ones also present.

Given the inherent noise in user generated, unstructured text and multiple languages in our data, we adopted the classical bag-of-words representation to encode product descriptions. First we concatenated the title and the description of the items. We explicitly avoided to repeat the title if this was already written at the beginning of the text. We filtered stopwords and extracted uni–grams and bigrams from text, and discarded all the entries that appear only once. Finally, the resulting bag-of-words was weighted using TF-IDF [118]. The final item representation is a sparse vector of length 1,099,425 with an average of 5.44 non-zero coordinates.

We considered other methods to extract features from unstructured text, e.g. distributed bag-of-words [92] and Language Modeling with RNNs [94]. However, the classical bag-of-words with TF-IDF was found to work better with our data. We attribute this to the noisiness of user generated content. The lack of English text and the presence of multiple languages prevented us from using pre-trained word representations, e.g. from Word2Vec.[2]

Experiments with adding an embedding layer between the features and the network resulted in worse performance, therefore, the classical bag-of-words/TF-IDF features were used as item representations and were used directly as the input of the RNNs.

4.3 Parallel RNNs

In this section, we describe the proposed parallel RNN (p-RNN) architectures that utilize item representations (features) for session modeling. A p-RNN consists of multiple RNNs, one for each representation/aspect of the item (e.g. one for ID, one for image and one for text). The hidden states of these networks are merged to produce the score for all items. We also introduce baseline architectures, naive approaches for using the different item representations.

As a basis, we take the best RNN setting from the session-based recommender described in Section 4.1.2. We used a single GRU layer without feedforward layers and the TOP1 pairwise loss function along with session-parallel mini-batching and output sampling. The input of the networks is the item ID of a session. The input in then translated into either:

[2]https://code.google.com/p/word2vec

a) a one-hot ID vector
b) a precomputed dense image feature vector
c) a sparse unigram + bigram text feature vector

The details on feature extraction are presented in Section 4.2. The proposed architectures use a subset of the above 3 item representations. The output is a score for every item indicating the likelihood of being the next item in the session. During training scores are compared to a one-hot vector created from the item ID of the next event in the session to compute the loss. In order to reduce computational costs, only the score of the target item and that of a small subset of "negative" items are computed during training.

4.3.1 Architectures

We devised the following architectures. For simplicity, we only present architectures with ID and image features; for text features one can proceed analogously as with image ones. The parallel architectures can also deal with ID, image and text features simultaneously. The architectures can be separated into two groups:

1. **Baseline architectures** (Figure 4.6):
   - **ID-only:** This architecture only uses the one-hot ID vectors and it is identical to the one described in Section 4.1.2. It serves as a baseline in our experiments.
   - **Feature-only:** The input of this variant is one of the content feature vectors (image or text). Otherwise it works similarly to the previous network.
   - **Concatenated input:** The easiest way to combine different item representations is to concatenate them. This network uses the concatenated representations as its input.
2. **p-RNN architectures** (Figure 4.7):
   - **Parallel:** The first parallel architecture trains one GRU network for each of the representations. Outputs are computed from the concatenation of the hidden layers of the subnets. This is equivalent to computing output scores separately, weighting them and applying through the final activation function. Training can be done in different ways (see training strategies in Section 4.3.2).
   - **Parallel shared-W:** This architecture differs from the previous one by having a shared hidden to output weight matrix. Scores

are not computed for each subnetwork separately. Instead, the weighted sum of the hidden states is multiplied by a single weight matrix to produce the output. Having a shared weight matrix greatly reduces the number of parameters and thus decreases training times and overfitting.
This model is also analogous to the pairwise model from context-aware factorization research. The hidden to output matrix is the item feature matrix. The hidden states are different representations of the session, i.e., different context dimensions.

- **Parallel interaction:** In this architecture, the hidden state of the item feature subnet(s) is multiplied by the hidden state of the ID subnet in an element-wise manner before computing the score of the subnet(s). Mixing different aspects of the session to compute item scores is analogous to context-aware preference modeling. For that task, [63] found that the *interaction model*, i.e. the sum of the user–item and user–context–item interaction to perform the best. This architecture mimics that model with the ID subnet being promoted to the primary representation of the session. The main difference to the context-aware task is that all of our representations are session representations and not (mostly) independent dimensions. Also note that contrary to the original interaction model, the output weight matrix (item feature matrix) is not shared in our model.

4.3.2 Training p-RNNs

Training parallel architectures is not trivial. Standard backpropagation across the whole architecture can produce suboptimal results due to different components of the architecture learning the same relations from the data. This can be avoided by pretraining some parts of the network and training the rest afterwards. This cycle can be done several times, motivated by the success of alternating methods like ALS for matrix factorization.
Note that, while the parameters of fixed networks remain unchanged, they still participate in the forward pass and they are only excluded from the backpropagation. We developed three training strategies for training p-RNNs, namely *simultaneous*, *alternating* and *residual* and *interleaved* training. We also provide the pseudo–code for alternating, residual and interleaved training in the case of training p-RNNs with ID and image features.

- **Simultaneous:** Every parameter of every subnet is trained simultaneously. It serves as the baseline.

Figure 4.6: *Baseline RNN architectures for feature inclusion. Architectures are presented with ID and image features only, but text features can be used in place of the image features as well. f() is the nonlinearity applied to the network output* (tanh *in our case). Architectures: ID-only; Concatenated input; Feature-only.*

Figure 4.7: *Parallel RNN architectures for feature inclusion. Architectures are presented with ID and image features only, but text features can be used in place of the image features as well. They can also be extended to have networks for ID, image and text features simultaneously. f() is the nonlinearity applied to the network output (*tanh *in our case). Architectures: Parallel; Parallel with interaction model; Parallel with shared weight matrix.*

	Sim.	Alt.	Res.	Int.
ID-only	✓			
Feature-only	✓			
Concatenated input	✓			
Parallel	✓	✓	✓	✓
Parallel shared-W	✓	✓	✓	✓
Parallel intmodel	✓	✓	✓	✓

Table 4.1: *Compatibilities between architectures and training strategies.*

- **Alternating** (Algorithm 1): Subnets are trained in an alternating fashion per epoch. For example, the ID subnet is trained in the first epoch, while the others are fixed; then we fix the ID subnet and train the image subnet for one epoch; and so on. The cycle restarts after each subnet was trained.

- **Residual** (Algorithm 2) Subnets are trained one after the other, on the residual error of the ensemble of the previously trained subnets. The cycle does not start over, but the individual training of a subnet is longer compared to the alternating method. For example, the ID subnet is trained for 10 epochs, then the image subnet is trained on the residual error of the ID subnet and so on.

- **Interleaving** (Algorithm 3): Alternating training per mini-batch. For each mini-batch of training examples, the first subnet is trained, the second subnet is trained on the residual error for the current mini-batch and so on. The more frequent alternation allows for a more balanced training across subnets without the drawbacks of the simultaneous training.

Not every training procedure is compatible with every architecture as we described above. Table 4.1 summarizes compatibilities.

4.4 Experiments

4.4.1 Datasets

The evaluation was done on two large proprietary datasets.

- VIDXL: was collected over a 2-month period from a Youtube-like video site, and contains video watching events having at least a predefined length. During the collection item-to-item recommendations

were displayed next to the featured video, generated by a selection of different algorithms.

- CLASS: consists of product view events of an online classified site. The site also had recommendations displayed with different algorithms during the collection period.

During the pre-processing of the raw event streams, we filtered out unrealistically long sessions as these are likely due to bot traffic. We removed sessions of one (single click) event because they are not useful for session-based recommendations and also removed items whose support is below five, as items with low support are not ideal for modeling. The sessions of the last day of each dataset are put into the test set. Each session is assigned to either the training or the test set, we do not split the data mid-session. We also filter items from the test dataset if they were not in the training set. This affects only a tiny fraction of the items. The datasets are summarized in Table 4.2.

Data	**Train set**		**Test set**		**Items**
	Sessions	Events	Sessions	Events	
VIDXL	17,419,964	69,312,698	216,725	921,202	712,824
CLASS	1,173,094	9,011,321	35,741	254,857	339,055

Table 4.2: *Properties of the datasets used in the experiments.*

4.4.2 Evaluation procedure

We evaluated the quality of the sequential next-event prediction task described in Section 2.6.1. Given an event of the session, we measured how well algorithms predict the next event of the session. The trained model is fed with the events of a session one after another and we check the rank of the selected item of the next event. The hidden state of the network is reset to zero after a session ends.

As recommender systems recommend only a few items at once, the relevant item should be amongst the first few items of the list. For this reason, we limited our analysis to the first 20 items in the recommendation list. Therefore, we used the following ranking metrics:

- *Recall@20*, or *hit-rate@20*, is the proportion of cases having the desired item amongst the top-20 items of all test cases. Recall does not consider the actual rank of the item as long as it is below 20. This is

an accurate model for certain practical scenarios where no recommendation is highlighted and their absolute order does not matter. Recall also usually correlates well with important online KPIs, such as click-through rate (CTR) [60].

- *MRR@20*, is the average of the reciprocal ranks of the desired items. The reciprocal rank is set to zero if the rank is above 20. MRR takes the rank of the item into account, which is important in cases where the order of recommendations matters (e.g. the lower ranked items are only visible after scrolling).

We compared the models described in Section 4.3 against the Item-kNN baseline, which recommends the most similar items to the current one in the session based on their session-level co-occurrence. The similarity between two items i and j is computed according to the formula

$$\text{sim}(i,j) = \frac{|S(i) \cap S(j)|}{\sqrt{|S(i)| + \lambda} \cdot \sqrt{|S(j)| + \lambda}} \tag{4.11}$$

where $S(z)$ returns the sessions in the training database where the item z occurs at least once, and λ is a shrinkage factor to avoid coincidental high similarities of rarely visited items. This baseline provides recommendations of the type "others who viewed this item also viewed these ones" and despite of its simplicity it is usually a strong baseline [79]. [61] shows that this baselines significantly outperforms other baselines such as general popularity, personalized popularity and latent factor models in scenarios of session-based similarity analogous to ours. In our experiments, we found the neighborhood size $k = 100$ and $\lambda = 20$ to be the optimal values.

4.4.3 Parameter tuning

We used random search [14] to tune the hyper-parameters of our models wrt. *Recall@20*. In random search, we used a separate validation set generated from the full training set using the same procedure described in Section 4.4. At test time, the best networks were retrained on the full training set and the final results were evaluated on the test set.

In parameter optimization, we considered single layer networks with 100 hidden units and TOP1 loss (defined in Section 4.1.2). Networks are trained with Adagrad [44], momentum [130] and Dropout regularization [128]. As in [61], single-layer RNNs always outperformed multi-layer ones, so we do not analyze them here.

Due to the size of the datasets, we optimized the hyper-parameters of the ID-only and Feature-only networks individually and used their optimal

Network	Loss	Mini-batch	Dropout	Learning rate	Momentum
ID-only	TOP1	100	0.2	0.05	0.2
Image-only	TOP1	100	0.1	0.1	0.0

Table 4.3: *Best parametrizations for the VIDXL dataset.*

values also with larger networks (i.e., with hidden size > 100) and in the parallel settings. While it is possible that better parameter configurations could be found by optimizing the parameters directly on the larger and more complex networks at the expense of a greater computational cost, this approach worked well in practice.

The hyper-parameters in common between ID-only and Feature-only networks are:

1. *learning rate:* the learning rate set in Adagrad;
2. *momentum:* momentum coefficient used in the training procedure;
3. *dropout:* dropout probability used to regularize the learning;
4. *batch size:* number of sessions in each parallel mini-batch.

Each feature network has some additional hyper-parameters depending on the nature of the features.

VIDXL dataset

Table 4.5 reports the optimal parameters for the ID and image network used in our experiments.

CLASS dataset

As we have introduced in Section 4.2.2, we considered different ways of using textual features with RNNs. Due to inherent noise in our multi-lingual, -generated textual data, the most complex language models like Word2Vec [92] did not perform well. The simple TF-IDF weighted uni-gram and bigram vectors outperformed the most complex language models by several orders of magnitude from the early stages of our analysis. For this reason, we did not consider complex language models in our final so-lution.

In Table 4.4, we present the values for Recall@20 (the metric used in the parameter tuning, as we described before) for different configurations of the Text-only RNN with TF-IDF features. The parameters of all the networks were optimized with the same random optimization procedure described before on the same validation set. Unigram and unigram+bigram features

Feature type	Size	Embedding	Dropout	Learning rate	Momentum	Recall@20
unigrams	200k	-	0.1	0.2	0.2	0.1812
unigrams	200k	200	0.1	0.1	0.2	0.1638
unigrams + bigrams	1M	-	0.2	0.2	0	**0.2089**
unigrams + bigrams	1M	200	0.2	0.05	0.3	0.1960

Table 4.4: *Best parametrizations for the text-only network on the CLASS dataset computed on the validation set. The optimal mini-batch size is 100 for all the entries in the table.*

are weighted with TF-IDF and then normalized to have unitary l^2-norm. In our experiments, the addition of bigrams produced a notable improvement in performance ($+15.28\%$ over unigram-only features) at the cost of a 5x larger set of features.

It is worth noting that the size of the input vector has a huge impact on the overall size of the Text-only network. In fact, the size of the input matrices of the GRU model grow linearly with the number of input features and, of course, with the hidden size. GRUs have 3 input-to-hidden matrices, thus a 2x longer feature vector requires 6x more memory for the input matrices that has to be allocated on the GPU (notice that GPUs nowadays have memory that is at least a order of magnitude *smaller* than RAM in workstations). We opted for GRUs over LSTMs also for space reasons (LSTMs have an additional input-to-hidden matrix).

We could partially handle this space complexity by using efficient sparse vector operations, since text feature vectors tend to be really sparse (on average 5.33 non-zero entries for unigram+bigram TF-IDF features). We also tried to convert the sparse feature representation into a smaller dense embedding vector in input to the GRU. By using embedding vectors we reduce the memory consumption of the model and the training times. We experimented with embeddings of different sizes ($100, 200, 500$). However, the addition of the embedding layer degrades notably the accuracy (-6.17% in its best configuration with unigram+bigram features and 200-dimensional embedding vectors), so we had to discard this option in favor of the original sparse high-dimensional feature vectors.

Table 4.5 reports the optimal parameters for the ID and text network used in our experiments.

Network	Loss	Mini-batch	Dropout	Learning rate	Momentum
ID-only	TOP1	100	0.3	0.05	0.3
Text-only	TOP1	100	0.2	0.2	0.0

Table 4.5: *Best parametrizations for the CLASS dataset.*

4.4.4 Session-based video recommendation with video thumbnails

We experimented with different architectures and training strategies described in Section 4.3 to see how image data can contribute to recommendation accuracy. Image feature were extracted with the procedure described in Section 4.2.1.

To speed up evaluation, we computed the rank of the relevant item compared to the 50,000 most supported items. While this evaluation methodology somewhat overestimates the rank and thus the evaluated metrics are a little bit higher, the comparison of the algorithms remains fair [12].

All the proposed network configurations were subject ti extensive study and experimentation on small-sized networks first. Then, the best configurations were promoted for experimentation on the large-sized networks. This step was necessary due to the size of the dataset that made full evaluation on large networks unpractical. The training times ranged from several hours for the small-size networks to days on the large-sized ones with cutting-edge GPUs.

Small-size networks

In this first experiment, we set the number of hidden units to 100 for the baseline architectures and 100 per subnetwork for p-RNNs (100+100 hidden units). The networks are trained for 10 epochs as there is no significant change in the training loss after that.

In Table 4.6, we summarize the results with different architectures and training methods in terms of Recall@20 and MRR@20 respectively. p-RNNs with 100+100 hidden units can easily outperform the ID-only network with 100 units. This can be likely due to the additional information source and the increase of the overall capacity of the network. Therefore, we also measured the accuracy of the ID-only network with 200 units. Note that this is a very strong baseline, because doubling the number of hidden units increases the capacity of the network ~ 4 times, while having 100+100 only doubles it. Also, the doubled capacity of p-RNN is split between two information sources, therefore it is clearly in disadvantage to even an RNN with doubled capacity. Nevertheless, we show that p-RNNs can often beat

Method	Recall@20	MRR@20
Item-kNN	0.6263 (+0.00%/-10.05%)	0.3740 (+0.00%/-3.63%)
ID only	0.6831 (+9.07%/-1.90%)	0.3847 (+2.86%/-0.88%)
ID only (200)	0.6963 (+11.18%/+0.00%)	0.3881 (+3.77%/+0.00%)
Feature only	0.5367 (-14.31%/-22.92%)	0.3065 (-18.05%/-21.03%)
Concatenated	0.6766 (+8.03%/-2.83%)	0.3850 (+2.94%/-0.80%)
Parallel (sim)	0.6765 (+8.02%/-2.84%)	0.4014 (+7.33%/+3.43%)
Parallel (alt)	0.6874 (+9.76%/-1.28%)	0.4331 (+15.80%/+11.59%)
Parallel (res)	**0.7028 (+12.21%/+0.93%)**	***0.4440 (+18.72%/+14.40%)***
Parallel (int)	***0.7040 (+12.41%/+1.11%)***	**0.4361 (+16.60%/+12.37%)**
Shared-W (sim)	0.6681 (+6.67%/-4.05%)	0.4007 (+7.14%/+3.25%)
Shared-W (alt)	0.6804 (+8.64%/-2.28%)	0.4035 (+7.89%/+3.97%)
Shared-W (res)	0.6425 (+2.59%/-7.73%)	0.3541 (-5.32%/-8.76%)
Shared-W (int)	0.6658 (+6.31%/-4.38%)	0.3715 (-0.67%/-4.28%)
Int. model (sim)	0.6751 (+7.79%/-3.04%)	0.3998 (+6.90%/+3.01%)
Int. model (alt)	0.6847 (+9.32%/-1.67%)	0.4104 (+9.73%/+5.75%)
Int. model (res)	0.6749 (+7.76%/-3.07%)	0.4098 (+9.57%/+5.59%)
Int. model (int)	0.6843 (+9.26%/-1.72%)	0.4040 (+8.02%/+4.10%)

Table 4.6: *Results on VIDXL, using image features extracted from thumbnails. The best performing model is typset in bold and the best performing p-RNN is italic. p-RNN architectures use 100+100 hidden units, others use 100. Performance gain over item-kNN and ID-only are shown in parentheses.*

this strong baseline as well, while they are typically better than the ID-only network with 100 units.

The ID-only RNN outperforms the item-kNN baseline and the other baselines by a large margin. The Recall for the ID-only RNN on this task is very high, therefore it is very hard for the more advanced architectures to significantly improve on this result.

The Feature-only RNN is worse than the ID-only network and even worse than item-kNN, demonstrating that the sequence of item features alone is not enough to model the session well. However, recommendations seemed appropriate from a manual look. The lower Recall and MRR can be attributed to several factors:

- The features extracted from the thumbnails may not concur with the ones that the users find interesting, therefore modeling the sessions only on these automatically extracted features is hard for the network.
- Offline evaluation probably underestimates its performance, because the user might have clicked on the item, had it been shown to her as a recommendation. The network selects an item with a similar picture to

the one the user clicked, but does not always select that exact item (see the example in Figure 4.5, many videos with the same content have similar feature representations, but only one is actually clicked by the user). This measurement bias is not present if items are represented as IDs, therefore it can be more accurate in an offline evaluation setup.

Using concatenated input of IDs and image features hardly differs from that of the ID-only network, since the stronger input dominates during the training. It is hard for a single GRU layer to handle two types of inputs at once, resulting in a performance very similar to that of the ID-only network. Adding item features using the naive approach has no observable benefits. Therefore, we propose to use p-RNNs instead.

Moving on to the proposed p-RNN architectures, one can see that several configurations outperform the strong ID-only baseline by large margins. Due to the originally high Recall of the network, these novel architectures mostly increase the MRR. This means that the they do not find more relevant items, but they rank them better. These results are also shown in Figure 4.11. Some considerations on each parallel setting follow.

- **Parallel** is the best performing architecture. With the naive simultaneous training it is much better than the strong ID-only baseline wrt. MRR, but slightly worse wrt. Recall. With simultaneous training, different components of the p-RNN learn the same relations from the data, thus the full capacity of the network is not leveraged. Therefore, we propose using alternative training strategies.

  All the alternative training methods are quite effective in this scenario, with alternative training being the best. The p-RNN with *residual training* outperforms the strong ID-only baseline by 14.40% in MRR, while achieving similar Recall. The improvement is even greater over the industry de facto item-kNN solution: 12.21% in Recall and 18.72% in MRR.

- **Parallel shared-W** is, conversely, the worst performing architecture. The *parallel shared-W* architecture constrains the output weight matrix to be shared between sub-networks in order to reduce overfitting. However, this does not necessarily translates in better prediction and recommendation performances, also because RNNs have other effective methods that fight overfitting, such as dropout, that might not work well in conjunction with this structure of the network.

  Additionally, the same adaptive learning rate method used to train the networks – Adagrad – might be responsible for the reduced perfor-

mance of this architecture. During training, Adagrad increasingly reduces the learning rate in a parameter-wise fashion, hence reducing the opportunity for a parameter to change as the training goes on. While discounting the learning rate has many beneficial effects on the convergence of the training procedure [44], it affects more severely parallel networks with a single, shared output matrix than ones with an output matrix per sub-network. Despite this, we found Adagrad to be the best adaptive learning rate method in our experiments.

- **Parallel interaction** achieves intermediate performance wrt. the other two architectures. Because output matrices are not shared between sub-networks, this architecture does not suffer from the same issues of the *shared-W* architecture. Still, the interaction model, by explicitly promoting the ID representation as the main representation of the session, seem to constraint excessively the modeling capabilities of the network. In comparison, the *parallel* architecture allows each sub-network to contribute freely to the final prediction, thus the interaction of the aspects modeled by each sub-network is learned directly from data.

The best performing architecture is the classic parallel one. With the naive simultaneous training it is significantly better than the strong ID-only baseline wrt. MRR, but slightly worse wrt. Recall. With simultaneous training, different components of the p-RNN learn the same relations from the data, thus the full capacity of the network is not leveraged. Therefore we propose using alternative training strategies.

Large-size networks

By increasing the number of hidden units, the capacity of the RNN increases, thus this parameter has a large effect on performance. However this parameter also obeys the law of diminishing returns. We found that results do not improve significantly above 1000 hidden units on this problem. We ran experiments with 1000 units on non-parallel and 500+500 units on the best performing p-RNN architecture (i.e. *parallel*) to confirm that adding item features can also benefit session modeling when increasing the network capacity and/or the number of epochs has diminishing returns.

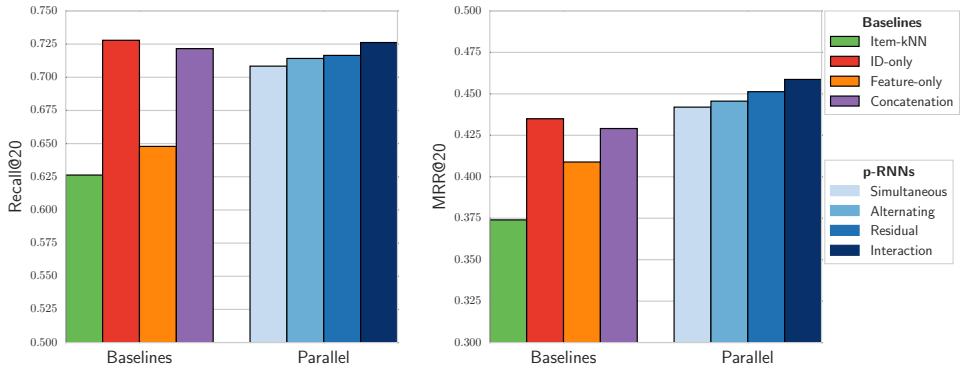
In Table 4.7 and Figure 4.8, we report the results in this setting. With more hidden units, the performance of all networks increases. Even the Feature-only network outperforms the item-kNN baseline as the capacity of the network is enough to leverage the information in the image features.

Network variant	Recall@20	MRR@20
Item-kNN	0.6263 (+0.00%/-13.96%)	0.3740 (+0.00%/-14.02%)
ID-only	0.7279 (+16.22%/+0.00%)	0.4350 (+16.31%/+0.00%)
Feature-only	0.6479 (+3.45%/-10.99%)	0.4089 (+9.33%/-6.00%)
Concatenated	0.7216 (+15.22%/-0.87%)	0.4291 (+14.73%/-1.36%)
Parallel (sim)	0.7084 (+13.11%/-2.68%)	0.4420 (+18.18%/+1.61%)
Parallel (alt)	0.7142 (+14.03%/-1.88%)	0.4456 (+19.14%/+2.44%)
Parallel (res)	0.7165 (+14.40%/-1.57%)	0.4513 (+20.67%/+3.75%)
Parallel (int)	***0.7262 (+15.95%/-0.23%)***	***0.4587 (+22.65%/+5.45%)***

Table 4.7: *Results on VIDXL, using image features extracted from thumbnails. The best performing model is typset in bold and the best performing p-RNN is italic. p-RNN architectures use 500+500 hidden units, others use 1000. Performance gain over item-kNN and ID-only are shown in parentheses.*

Still, recommendation based on item IDs is more effective, for the same reasons of the previous experiment.

A part from this case, the relation between the results is similar to that of the previous experiments. This further underpins that p-RNNs with alternative training strategies are vital for efficiently incorporating item features into learning session models. Further increasing the number of hidden units and/or the number of epochs did not increase the performance of any network with large margins, but p-RNN architectures notably outperform the ID-only network with more than 2 times larger capacity in terms of MRR (5.45% with interleaving training) and have similar Recall. In summary,



(a) *Recall* **(b)** *MRR*

Figure 4.8: *Recall and MRR for session-based recommendation with image features on the VIDXL dataset (large-size networks). Note: we represent only the best p-RNN setting (parallel) obtained from the experiments with small-size networks.*

parallel architectures with the proposed training strategies can notably increase performance, even when increasing the capacity of the network has diminishing returns. In other words, adding additional data sources (image features) can increase the accuracy and ranking in recommendations beyond the maximum achievable just from item IDs in this scenario.

4.4.5 Session-based classified advertisement recommendation with product descriptions

We experimented with the different architectures and training strategies to see how textual features extracted from product descriptions can contribute to the recommendation accuracy in the scenario of classified advertisement recommendation. The details on feature extraction from textual data are reported in Section 4.2.2. As in the experiments with image features, we experimented with small-size networks first. The best architectures in this setting were successively promoted for experimentation with the large-size networks.

Small-size networks

We used the same setting used for image features. We set the number of hidden units to 100 for the baseline architectures and to 100 for each subnetwork for p-RNNs (100+100). In this case, we have not restricted the evaluation to a subset of the most popular items in the dataset and all items instead. This partly explains the overall lower Recall and MRR values that we obtained in this scenario.

The results for Recall@20 and MRR@20 are reported in Table 4.8. Similar considerations to the case with image features apply also to this scenario. ID-only RNNs outperform the Item-kNN by a large margin. Most p-RNNs configurations outperform Item-KNN in terms of MRR and achieve similar results in terms of Recall. However, none of them can outperform the ID-only network with 200 hidden units. The ID-only network can make better usage of the additional capacity wrt. parallel networks with 100+100 units.

A reason for this can be that learning from sparse high-dimensional textual feature vectors is considerably much harder than learning from item IDs. In fact, the Feature-only network is considerably worse than the ID-only one and Item-kNN. The lower performances can be attributed to:

- The sparsity in the input features requires networks with larger capacity to be exploited properly and to learn useful relationship between

Method	Recall@20	MRR@20
Item-kNN	0.2387 (+0.00%/-12.34%)	0.0839 (+0.00%/-18.57%)
ID only	0.2478 (+3.82%/-8.99%)	0.0949 (+13.11%/-7.89%)
ID only (200)	**0.2723 (+14.08%/+0.00%)**	**0.1030 (+22.80%/+0.00%)**
Feature only	0.2148 (-9.99%/-21.09%)	0.0751 (-10.52%/-27.13%)
Concatenated	0.2171 (-9.03%/-20.25%)	0.0789 (-5.97%/-23.43%)
Parallel (sim)	0.2303 (-3.53%/-15.43%)	0.0837 (-0.18%/-18.71%)
Parallel (alt)	0.2557 (+7.13%/-6.09%)	0.0914 (+8.93%/-11.30%)
Parallel (res)	*0.2704 (+13.30%/-0.68%)*	*0.1010 (+20.37%/-1.98%)*
Parallel (int)	0.2572 (+7.75%/-5.55%)	0.0940 (+12.04%/-8.76%)
Shared-W (sim)	0.2274 (-4.72%/-16.48%)	0.0827 (-1.41%%/-19.71%)
Shared-W (alt)	0.2463 (+3.21%/-9.52%)	0.0904 (+7.82%/-12.20%)
Shared-W (res)	0.2409 (+0.91%/-11.54%)	0.0908 (+8.23%/-11.86%)
Shared-W (int)	0.2310 (-3.24%/-15.18%)	0.0836 (-0.36%/-18.86%)
Int. model (sim)	0.2298 (-3.72%/-15.60%)	0.0835 (-0.45%/-18.94%)
Int. model (alt)	0.2388 (+0.05%/-12.29%)	0.0875 (+4.27%/-15.09%)
Int. model (res)	0.2209 (-7.43%/-18.85%)	0.0797 (-5.03%/-22.66%)
Int. model (int)	0.2390 (+0.14%/-12.22%)	0.0885 (+5.50%/-14.09%)

Table 4.8: *Results on CLASS, using image features extracted from thumbnails. The best performing model is typset in bold and the best performing p-RNN is italic. p-RNN architectures use 100+100 hidden units, others use 100. Performance gain over item-kNN and ID-only are shown in parentheses.*

subsequent items. This fact can be easily verified by using large-sized networks.

- Similarly to image features, accuracy metrics might be underestimating the performance of the network. In fact, there might exist many items having similar descriptions, hence very similar feature vectors. The network selects an item with a similar description to the one that is actually clicked by the user, but does not always selects the item the user actually clicked. This inherent measurement bias affects less the ID-only network, therefore it can be more accurate in an offline setup.

Using concatenated input IDs and textual features now performs similar to the Feature-only network. The input of the network is the concatenation of the sparse one-hot vector for the item ID (of size $\sim 340k$) and the sparse textual feature vector (of size $\sim 1M$). Textual features are now the dominating term in the input, consequently performances are really close to the Feature-only one. Simple concatenation is not viable with sparse features and IDs, therefore we must rely on parallel architectures to exploit properly the two sources of information.

Training info	Recall@20	MRR@20
Item-kNN baseline	0.2387 (+0.00%/-16.22%)	0.0839 (+0.00%/-21.00%)
ID-only, 10 epochs	0.2849 (+19.35%/+0.00%)	0.1062 (+26.58%/+0.00%)
Feature-only	0.2397 (+0.42%/-15.87%)	0.0937 (+11.68%/-11.77%)
Conc. input	0.2844 (+19.15%/-0.18%)	0.1029 (+22.65%/-3.11%)
Parallel (sim)	0.2741 (+14.83%/-3.79%)	0.1019 (+21.45%/-4.05%)
Parallel (alt)	0.2877 (+20.53%/+0.98%)	0.1096 (+30.63%/+3.20%)
Parallel (res)	***0.2946 (+23.42%/+3.40%)***	***0.1119 (+33.37%/+5.37%)***
Parallel (int)	0.2854 (+19.56%/+0.18%)	0.1058 (+26.10%/-0.38%)

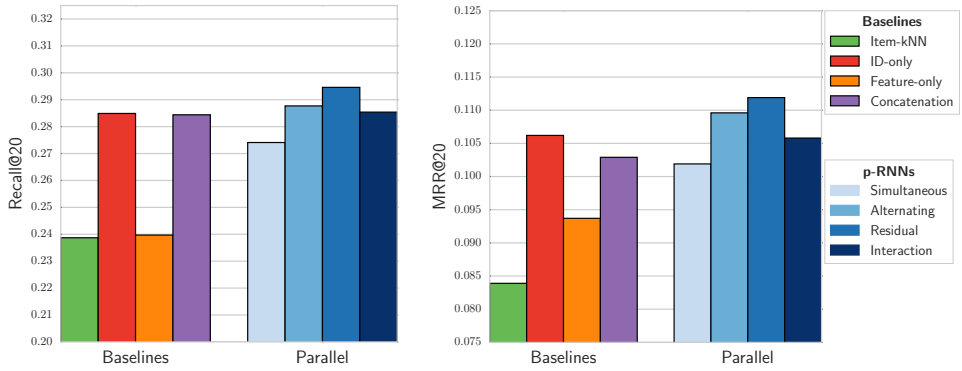
Table 4.9: *Results on CLASS, using textual features extracted from product descriptions. The best performing model is typset in bold and the best performing p-RNN is italic. p-RNN architectures use 500+500 hidden units, others use 1000. Performance gain over item-kNN and ID-only are shown in parentheses.*

The comparison between p-RNNs is in line with the previous findings with image features. The *parallel* setting clearly performs better than others both in Recall and MRR by large margins. Therefore, we consider only it in the experiments with large-size networks.

Large-size networks

We again used 1000 hidden units for the baselines architectures and 500+500 for p-RNNs. The results shown in Table 4.9 are in line with that of the earlier experiments with image features. The text only network outperforms by a large margin the item-kNN baseline in terms of MRR. This confirms that textual features can be effectively exploited to generate better rankings when the network has sufficient capacity. However, it falls short when compared with the ID-only network. This confirms that text features alone are not enough. With concatenated input, the network performs similarly to the ID-only network analogously thanks to the larger capacity of the text sub-network.

The alternative training strategies are of crucial importance when training p-RNNs. As in the experiment with image features, the simultaneous training is clearly sub-optimal wrt. recommendation accuracy. The classic p-RNN with alternative training strategies outperform both the ID-only RNN and item-kNN in *both* Recall and MRR by a large margin. Residual training proved to be the best strategy in this experiment with $\sim 3\%$ improvement in Recall and $\sim 5\%$ in MRR over the ID-only network. Note that further increasing the number of hidden units or number of epochs for the baseline RNNs did not improve the results any further. Thus, using



(a) *Recall* **(b)** *MRR*

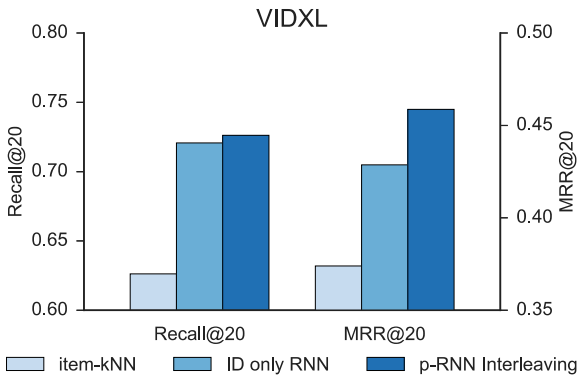
Figure 4.9: *Recall and MRR for session-based recommendation with image features on the CLASS dataset (large-size networks). Note: we represent only the best p-RNN setting (parallel) obtained from the experiments with small-size networks.*

text based item features in p-RNNs with proper training can also increase recommendation accuracy beyond what is achievable from IDs only.

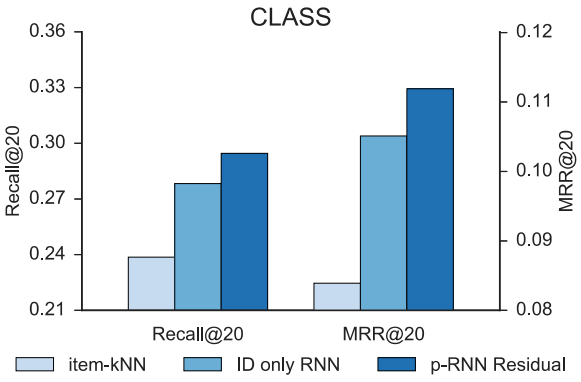
4.5 Conclusions

Recommendation in session-based scenarios is an extremely hard task. It is much more complex to gather information about the users' tastes and interests, due to the complete absence of the historical profile of the user. In this scenario all the classical approaches to recommendation beside simple item-kNN fall short, and one needs to find new solutions in order to effectively learn the users' interest and intent out of the handful of interactions she may perform in a session. The sequence of user interactions certainly constitutes a crucially important source of information to this purpose. In fact, it is known that patterns in the sequence of items IDs can be extracted from the logs of activities of users and exploited to generate next-item recommendations. Beside the sequence of item IDs, also item features provide valuable information on the interests of the user in a session.

In this chapter, we examined the use of item features (image data and text) in RNN-based session modeling to improve session-based recommendations. We focused on next-item recommendation in two real-world scenarios with user-generated content, namely video recommendation and classified advertisement recommendation. We used features that are automatically extracted from unstructured noisy data (video thumbnails and multilingual product descriptions). This is in line with real-life experience



(a) *VIDXL dataset*



(b) *CLASS dataset*

Figure 4.10: *Comparing best performing p-RNN against the ID-only RNN and item-kNN.*

in which high-quality human-curated metadata is available only for a small fraction of items, whereas the large majority of items are represented only by their unstructured content. Despite the inherent challenges of this setting, we discovered that RNN-based session models can effectively exploit item features to improve the recommendation quality, after some needed adaptations to handle the additional sources of information properly. We pointed out that item features are not enough to properly model the session and combining multiple data sources efficiently is not trivial. We proposed p-RNN architectures that can leverage the added value of multiple item representations. We devised alternative training strategies (alternating, residual and interleaving training) that fit these architectures better than backpropagating the error through the whole network. The proposed architectures and training methods outperformed both RNNs with ID-only input and item-kNN – the de-facto industry standard for this problem – by large margins. Finally, we showed that by using p-RNNs with alternative

training, recommendation accuracy can be increased beyond the maximum that is achievable by RNNs with ID-only input by increasing their capacity and/or the number of training epochs. We demonstrate the power of the proposed solution (500 units per subnet) by comparing it to item-kNN and the ID-only network with 1000 hidden units on Figure 4.10.

Notwithstanding the positive results with item features, the most significant improvements are relative MRR, hence to the capability of bringing interesting item up in the recommended ranked list. The improvements in Recall are mostly marginal, i.e. the capability of retrieving new interesting items for the user is almost unchanged. However, offline evaluation may shadow part of the improvements since feature-based networks tend to recommend items having the right characteristics for the user but that may have not been actually clicked by her. An online study may help in clarifying this point.

Item features can also be used to recommend new and cold-items. This is a desirable feature for domains where items have a high turn-over rates (like in classified advertisement for example), in which items added and removed from the pool of the available ones at faster pace that what can be effectively covered by retraining traditional ID-based methods.

ALGORITHM 1: Alternating training (training order: Id, Img)

Input : Network parameters $\Theta = \{\Theta_{Id}, \Theta_{Img}\}$; Final non-linearity g; Loss function L; Training dataset D; Training epochs N

/* $\hat{y} = g\left(W_{Id}h_{Id} + W_{Img}h_{Img} + b_y\right)$ */

foreach *training epoch in* $\{1..N\}$ **do**

  /* 1) fix Θ_{Img}, optimize $L(y, \hat{y}|D; \Theta_{Id})$ */

  /* 2) fix Θ_{Id}, optimize $L(y, \hat{y}|D; \Theta_{Img})$ */

end

ALGORITHM 2: Residual training (training order: Id, Img)

Input : Network parameters $\Theta = \{\Theta_{Id}, \Theta_{Img}\}$; Final non-linearity g; Loss function L; Training dataset D; Training epochs N

/* 1) Optimize the Id-network only */

/* $\hat{y}_{Id} = g\left(W_{Id}h_{Id} + b_y\right)$ */

/* Optimize $L(y, \hat{y}_{Id}|D; \Theta_{Id})$ for N epochs */

/* 2) Fix the Id-network, optimize the Img-network only */

/* $\hat{y}_{Img} = g\left(W_{Id}h_{Id} + W_{Img}h_{Img} + b_y\right)$ */

/* Optimize $L(y, \hat{y}_{Img}|D; \Theta_{Img})$ for N epochs */

ALGORITHM 3: Interleaved training (training order: Id, Img)

Input : Network parameters $\Theta = \{\Theta_{Id}, \Theta_{Img}\}$; Final non-linearity g; Loss function L; Training dataset D; Training epochs N

/* $\hat{y} = g\left(W_{Id}h_{Id} + W_{Img}h_{Img} + b_y\right)$ */

foreach *training epoch in* $\{1..N\}$ **do**

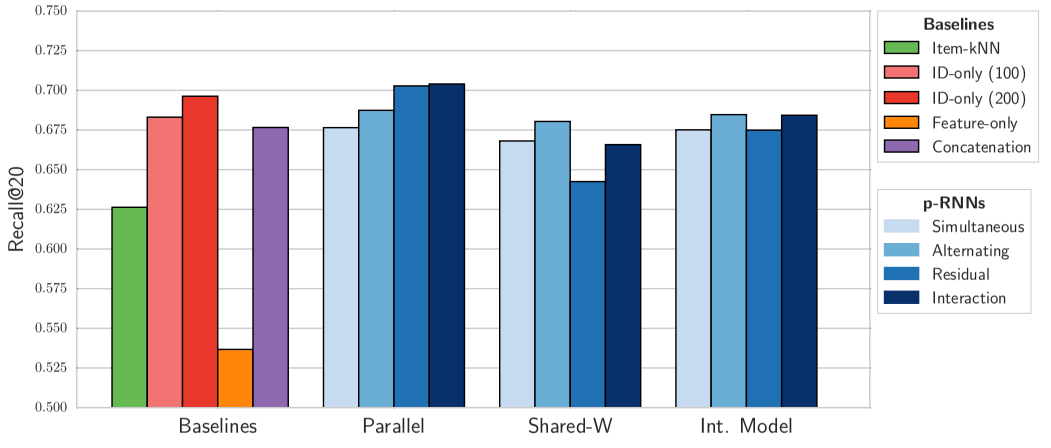
  foreach *mini-batch* $\mathbf{x}_B \in D$ **do**

    /* 1) fix (h_{Img}, Θ_{Img}), optimize for $L(y, \hat{y}|\mathbf{x}_B; \Theta_{Id})$ */

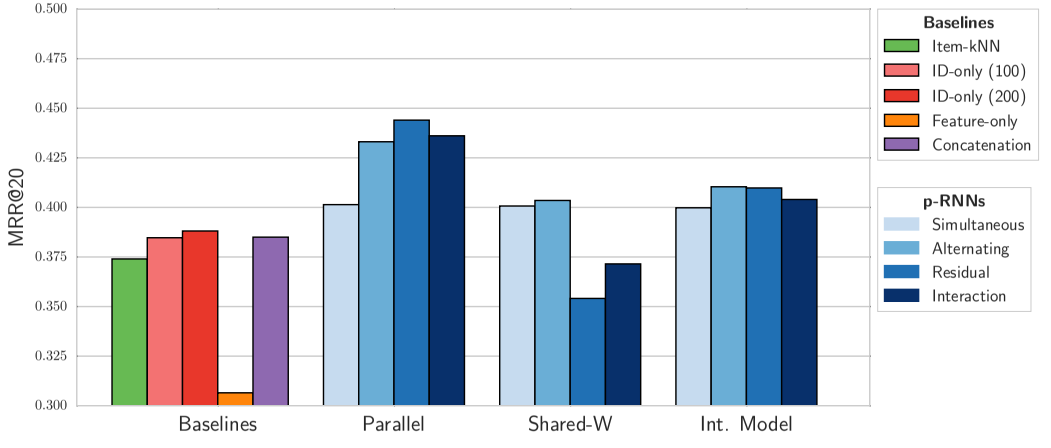
    /* 2) fix (h_{Id}, Θ_{Id}), optimize for $L(y, \hat{y}|\mathbf{x}_B; \Theta_{Img})$ */

  end

end

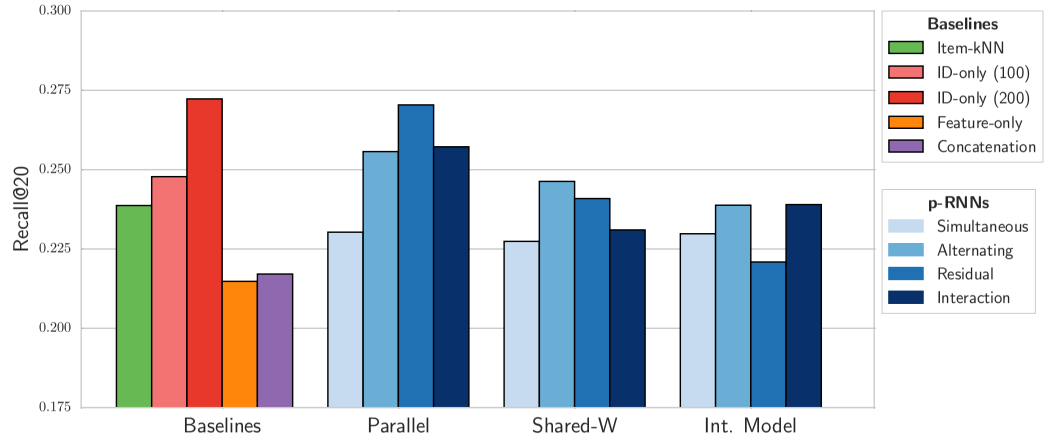


(a) *Recall*

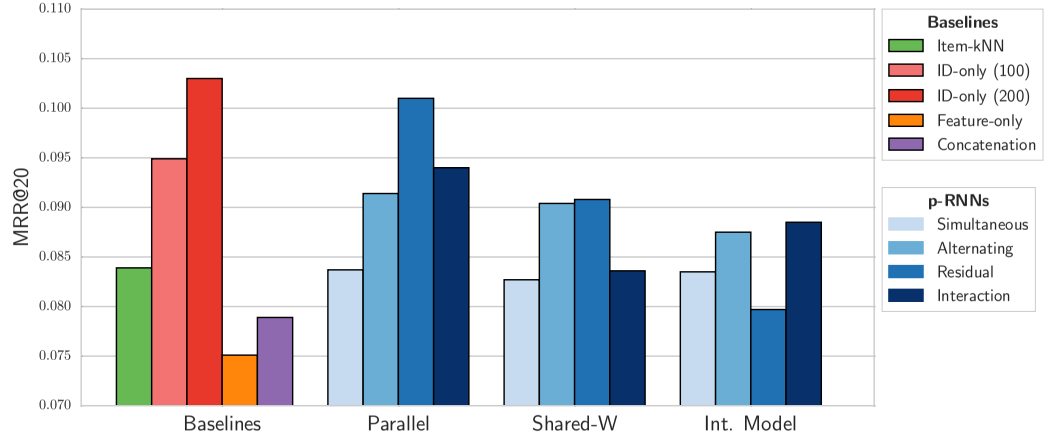


(b) *MRR*

Figure 4.11: *Recall and MRR for session-based recommendation with image features on the VIDXL dataset (small-size networks).*



(a) *Recall*



(b) *MRR*

Figure 4.12: *Recall and MRR for session-based recommendation with image features on the CLASS dataset (small-size networks).*

CHAPTER 5

Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks

In many online systems where recommendations are applied, interactions between a user and the system are organized into sessions. A session is a group of interactions that take place within a given time frame. Sessions from a user can occur on the same day, or over several days, weeks, or months. A session usually has a goal, such as finding a good restaurant in a city, or listening to music of a certain style or mood.

Providing recommendations in these domains poses unique challenges that until recently have been mainly tackled by applying conventional recommender algorithms [72] on either the last interaction or the last session (session-based recommenders). Recurrent Neural Networks (RNN's), as we have analyzed in the previous chapter, can be used for the purpose of session-based recommendations outperforming item-based methods by 15% to 30% in terms of ranking metrics. In *session-based recommenders*, recommendations are provided based solely on the interactions in the current user session, as user are assumed to be anonymous. But in many of these systems there are cases where a user might be logged-in (e.g. music

streaming services) or some form of user identifier might be present (cookie or other identifier). In these cases it is reasonable to assume that the user behavior in past sessions might provide valuable information for providing recommendations in the next session.

The standard approach in recommender systems for recommendation in a setting where user profiles are available is to use some kind of standard collaborative filtering approach such as memory-based methods, item-kNN, etc. In the case of session-based recommendations this would though make it difficult or impossible to incorporate the current session-information into the model. Thus information from more than one click into the past is typically ignored. On the other hand the simple RNN algorithm for session-based recommendations would ignore information from the users past sessions. A simple way of incorporating past user session information in session-based algorithm would be to simply concatenate past and current user sessions. While this seems like a reasonable approach, we will see in the experimental section that this does not yield the best results.

In this chapter we describe a novel algorithm based on RNN's that can deal with both cases: (i) *session-aware recommenders*, when user identifiers are present and propagate information from the previous user session to the next, thus improving the recommendation accuracy, and (ii) *session-based recommenders*, when there are no past sessions (i.e., no user identifiers). The algorithm is based on a Hierarchical RNN where the hidden state of a lower-level RNN at the end of one user session is passed as an input to a higher-level RNN which aims at predicting a good initialization (i.e., a good context vector) for the hidden state of the lower RNN for the next session of the user.

We evaluate the Hierarchical RNN's on two datasets from industry comparing them to the plain session-based RNN and to item-based collaborative filtering. Hierarchical RNN's outperform both alternatives by a healthy margin.

This work was presented as a full-paper at the *ACM Recsys 2017* conference [109].

The outline of this chapter is the following. In Section 5.1 we present the background and related works. In Section 5.2 we present the HRNN model and the improved mini-batch learning for personalized session-based recommendation. In Section 5.3 we present the experiments and discuss about the results. In Section 5.4 we present the conclusions and the possible future developments.

5.1 Background and Related works

The existing works in session-based and session-aware recommendation and Recurrent Neural Network have been already extensively discussed in Chapter 2 and in Section 4.1. Therefore, we focus here on the differences between the existing works in literature that target session-aware recommender systems and the approach presented in this chapter.

Jannach et al. [66],propose to use non-contextualized baseline strategies to learn long-term user preferences and combine them with ad-hoc contextualization strategies to adapt recommendations to the actions performed by the user in the most recent sessions. A similar approach is used by [67] for automated playlist generation, in which tracks are first ranked according to the long-term user interests, and successively re-ranked according to the characteristics of the current listening session or playlist. Instead of using ad-hoc contextualization strategies for short-term recommendation, our HRNN method instead seamlessly integrates both long-term and short-term user preferences into a single session-aware model. The short-term model can be seen as an "enhanced" (personalized) session-based model, that models the activity of the user in the current session and, at the same time, leverages the historical activity of the user to decide what to recommend next.

Natarajan et al. [99] propose to cluster users according to their long-term behavior first, and then cluster-level transition models are used for contextual adaptation at session-level. Finally, Xiang et al. [143] suggest to model historical user activity and session-level user interactions into a bipartite graph that is later browsed with random walks. While both methods can jointly model both long-term and short-term favorites, they do not explicitly model the sequentiality of the events at session level and do not represent the evolution of the user preferences over time, as our HRNN model instead does.

5.2 Model

In this section we describe the proposed Hierarchical RNN (HRNN henceforth) model for personalized session-based recommendation.

5.2.1 Session-based RNN

Our model is based on the session-based Recurrent Neural Network (RNN henceforth) model presented in Section 4.1.2. To ease the explanation of

the proposed HRNN model, we briefly describe RNN with a new notation similar to the one used in [127].

As we have already seen in the previous chapter, the session-based RNN model takes as input the current item ID in the session and outputs a score for each item representing the likelihood of being the next item in the session. Formally, for each session $S_m = \{i_{m,1}, i_{m,2}, ..., i_{m,N_m}\}$, RNN computes the following session-level representation

$$s_{m,n} = GRU_{ses}\left(i_{m,n}, s_{m,n-1}\right),\ n = 1, ..., N_m - 1 \tag{5.1}$$

where GRU_{ses} is the *session-level* GRU and $s_{m,n}$ its hidden state at step n, being $s_{m,0} = 0$ (the null vector), and $i_{m,n}$ is the one-hot vector of the current item ID..

The output of the RNN is a score $\hat{r}_{m,n}$ for every item in the catalog indicating the likelihood of being the next item in the session (or, equivalently, its relevance for the next step in the session)

$$\hat{r}_{m,n} = g\left(s_{m,n}\right),\ n = 1, ..., N_m - 1 \tag{5.2}$$

where $g\left(\cdot\right)$ is a non-linear function like softmax or tanh depending on the loss function. During training, scores are compared to a one-hot vector of the next item ID in the session to compute the loss. The network can be trained with several ranking loss functions such as cross-entropy, BPR [111] and TOP1 [61]. In this work, the TOP1 loss always outperformed other ranking losses, so we consider only it in the rest of the chapter. RNN is trained efficiently with session-parallel mini-batches. For the exhaustive explanation of all the ranking loss functions and the session-parallel mini-batch mechanism, please refer to Section 4.1.2.

It is worth noting that, since user-identifiers are unknown in pure session-based scenarios, in the session-parallel mini-batch mechanism there are good chances that negative samples will be "contaminated" by actual positive, i.e. items the user may interact with in her other sessions. This because the negative samples for one event in a session are taken from the positive items of the other sessions in the mini-batch, but no constraint is posed on which users the other sessions in the mini-batch must belong to. Such undesired effect can be effectively mitigated when user IDs are known, as we will describe in the following section.

5.2.2 Personalized Session-based HRNN

Our HRNN model builds on top of RNN by: (i) adding an additional GRU layer to model information across user sessions and to track the evolution

Figure 5.1: *Graphical representation of the proposed Hierarchical RNN model for personalized session-based recommendation. The model is composed of an hierarchy of two GRUs, the session-level GRU (GRU_{ses}) and the user-level GRU (GRU_{usr}). The session-level GRU models the user activity within sessions and generates recommendations. The user-level GRU models the evolution of the user across sessions and provides personalization capabilities to the session-level GRU by initializing its hidden state and, optionally, by propagating the user representation in input.*

of the user interests over time; (ii) using a powerful user-parallel mini-batch mechanism for efficient training.

Architecture

Beside the session-level GRU, our HRNN model adds one *user-level* GRU (GRU_{usr}) to model the user activity across sessions.

Figure 5.1 shows a graphical representation of HRNN. At each time step, recommendations are generated by GRU_{ses}, as in RNN. However, when a session ends, the user representation is updated. When a new session starts, the hidden state of GRU_{usr} is used to initialize GRU_{ses} and, optionally, propagated in input to GRU_{ses}.

Formally, for each user u with sessions $C^u = \{S_1^u, S_2^u, ..., S_{M_u}^u\}$, the user-level GRU takes as input the session-level representations $s_1^u, s_2^u, ..., s_{M_u}^u$, being $s_m^u = s_{m,N_m-1}^u$ the last hidden state of GRU_{ses} of each user session S_m^u, and uses them to update the user-level representation c_m^u. Henceforth we drop the user superscript u to unclutter notation. The user-level representation c_m is updated as

$$c_m = GRU_{usr}\left(s_m, c_{m-1}\right),\ m = 1, ..., M_u \tag{5.3}$$

where $c_0 = 0$ (the null vector). The input to the user-level GRU is connected to the last hidden state of the session-level GRU. In this way, the user-level GRU can track the evolution of the user across sessions and, in turn, model the dynamics user interests seamlessly. Notice that the user-level representation is kept *fixed* throughout the session and it is updated only when the session ends.

The user-level representation is then used to initialize the hidden state of the session-level GRU. Given c_m, the initial hidden state $s_{m+1,0}$ of the session-level GRU for the following session is set to

$$s_{m+1,0} = \tanh\left(W_{init} c_m + b_{init}\right) \tag{5.4}$$

where W_{init} and b_{init} are the initialization weights and biases respectively. In this way, the information relative to the preferences expressed by the user in the previous sessions is transferred to the session-level. Session-level representations are then updated as follows

$$s_{m+1,n} = GRU_{ses}\left(i_{m+1,n}, s_{m+1,n-1}\left[, c_m\right]\right),\ n = 1, ..., N_{m+1} - 1 \tag{5.5}$$

where the square brackets indicate that c_m can be *optionally* propagated in input to the session-level GRU.

The model is trained end-to-end using back-propagation [115]. The weights of GRU_{usr} are updated only between sessions, i.e. when a session ends and when the forthcoming session starts. However, when the user representation is propagated in input to GRU_{ses}, the weights of GRU_{usr} are updated also within sessions even if c_m is kept fixed. We also tried with propagating the user-level representation to the final prediction layer (i.e., by adding term c_m in Equation 5.2) but we always incurred into severe degradation of the performances, even wrt. simple session-based RNN. We therefore discarded this setting from this discussion.

Note here that the GRU_{usr} does not simply pass on the hidden state of the previous user session to the next but also learns (during training) how user sessions evolve during time. We will see in the experimental section that this is crucial in achieving increased performance. In effect GRU_{usr} computes and evolves a user profile that is based on the previous user sessions, thus in effect personalizing the GRU_{ses}. In the original RNN, users who had clicked/interacted with the same sequence of items in a session would get the same recommendations; in HRNN this is not anymore the case, recommendations will be influenced by the the users past sessions as well.

In summary, we considered the following two different HRNN settings, depending on whether the user representation c_m is considered in Equation 5.5:

- *HRNN Init*, in which c_m is used only to initialize the representation of the next session.
- *HRNN All*, in which c_m is used for initialization and propagated in input at each step of the next session.

In *HRNN Init*, the session-level GRU can exploit the historical preferences along with the session-level dynamics of the user interest. *HRNN All* instead enforces the usage of the user representation at session-level at the expense of a slightly greater model complexity. As we will see, this can lead to substantially different results depending on the recommendation scenario.

Learning

For the sake of efficiency in training, we have edited the session-parallel mini-batch mechanism described in [62] to account for user identifiers during training (see Figure 5.2). We first group sessions by user and then sort session events within each group by time-stamp. We then order users at

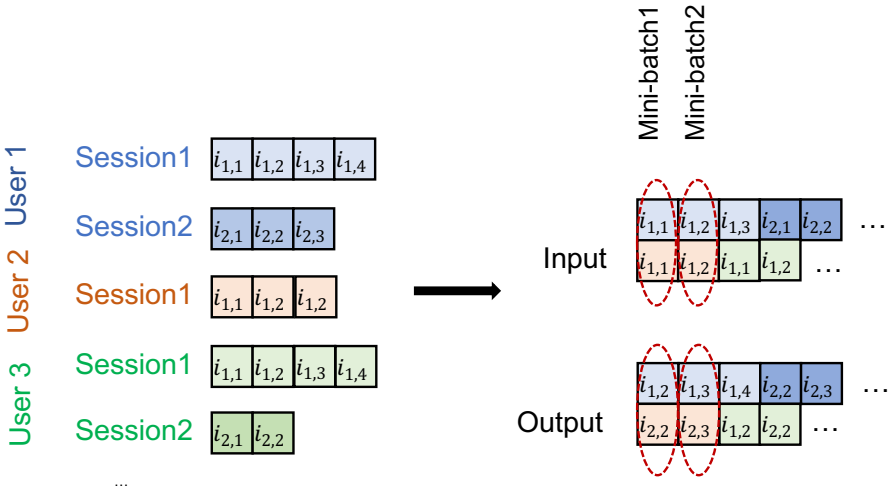


Figure 5.2: *User-parallel mini-batches for mini-batch size 2.*

random. At the first iteration, the first item of the first session of the first B users constitute the input to the HRNN; the second item in each session constitute its output. The output is then used as input for the next iteration, and so on. When a session in the mini-batch ends, Equation 5.3 is used to update the hidden state of GRU_{usr} and Equation 5.4 to initialize the hidden state of GRU_{ses} for the forthcoming session, if any. When a user has been processed completely, the hidden states of both GRU_{usr} and GRU_{ses} are reset and the next user is put in its place in the mini-batch.

With user-parallel mini-batches we can train HRNNs efficiently over users having different number of sessions and sessions of different length. Moreover, this mechanism allows to sample negative items in a user-independent fashion, hence reducing the chances of 'contamination' of the negative samples with actual positive items. The sampling procedure is still popularity-based, since the likelihood for an item to appear in the mini-batch is proportional to its popularity. Both properties are known to be beneficial for pairwise learning with implicit user feedback [110].

5.3 Experiments

In this section we describe our experimental setup and discuss in depth the results.

5.3.1 Datasets

We used two datasets for our experiments. The first is the XING [1] Recsys Challenge 2016 dataset [1] that contains interactions on job postings for 770k users over a 80-days period. User interactions come with time-stamps and interaction type (click, bookmark, reply and delete). We named this dataset XING. The second dataset is a proprietary dataset from a Youtube-like video-on-demand web site. The dataset tracks the videos watched by 13k users over a 2-months period. Viewing events lasting less than a fixed threshold are not tracked. We named this dataset VIDEO.

We manually partitioned the interaction data into sessions by using a 30-minute idle threshold. For the XING dataset we discarded interactions having type 'delete'. We also discarded repeated interactions of the same type within sessions to reduce noise (e.g. repeated clicks on the same job posting within a session). We then preprocessed both datasets as follows. We removed items with support less than 20 for XING and 10 for VIDEO since items with low support are not optimal for modeling. We removed sessions having < 3 interactions to filter too short and poorly informative sessions, and kept users having ≥ 5 sessions to have sufficient cross-session information for proper modeling of returning users.

The test set is build with the last session of each user. The remaining sessions form the training set. We also filtered items in the test set that do not belong to the training set. This partitioning allows to run the evaluation over users having different amounts of historical sessions in their profiles, hence to measure the recommendation quality over users having different degrees of activity with the system (see Section 5.3.4 for an in-depth analysis). We further partitioned the training set with the same procedure to tune the hyper-parameters of the algorithms. The characteristics of the datasets are summarized in Table 5.1.

5.3.2 Baselines and parameter tuning

We compared our HRNN model against several baselines, namely Personal Pop (PPOP), Item-KNN, RNN and RNN Concat:

- Personal Pop (PPOP) recommends the item with the largest number of interactions by the user.
- Item-KNN computes an item-to-item cosine similarity based on the co-occurrence of items within sessions.

[1] https://www.xing.com/en

Dataset	XING	VIDEO
Users	11,479	13,717
Items	59,297	19,218
Sessions	89,591	133,165
Events	546,862	825,449
Events per item†	6/9.3/14.4	9/43.0/683.7
Events per session†	4/6.1/5.5	4/6.2/8.2
Sessions per user†	6/7.8/4.8	7/9.8/7.8
Training - Events	488,576	745,482
Training - Sessions	78,276	120,160
Test - Events	58,286	79,967
Test - Sessions	11,315	13,605

Table 5.1: *Main properties of the datasets (†median/mean/std).*

- RNN adopts the same model described in [61]. This model uses the basic GRU with a TOP1 loss function and session-parallel minibatching: sessions from the same user are fed to the RNN independently from each other.

- RNN Concat is the same as RNN, but sessions from the same user are concatenated into a single session[2].

We optimize the neural models for TOP1 loss using AdaGrad [44] with momentum for 10 epochs. Increasing the number of epochs did not significantly improve the loss in all models. We used dropout regularization [128] on the hidden states of RNN and HRNN. We applied dropout also to GRU_{ses} initialization for HRNN (Equation 5.4). We used single-layer GRU networks in both levels in the hierarchy, as using multiple layers did not improve the performance. In order to assess how the capacity of the network impacts the recommendation quality, in our experiments we considered both *small* networks with 100 hidden units per GRU layer and *large* networks with 500 hidden units per GRU layer.

We tuned the hyper-parameters of each model (baselines included) on the validation set using random search [14]. The hyper-parameters used in our experiments on the XING and VIDEO datasets are reported in Table 5.2 and Table 5.3 respectively. Dropout probabilities for HRNNs are relative to the user-level GRU, session-level GRU and initialization in this order. The optimal neighborhood size for Item-KNN is 300 for both datasets.

[2]We experimented with a variant of this baseline that adds a special 'separator item' to enforce the separation between sessions; however, it did not show better performance in our experiments.

Model	Batch size	Dropout	Learning rate	Momentum
RNN (*small*)†	100	0.2	0.1	0.1
RNN (*large*)†	100	0.2	0.1	0.0
HRNN All (*small*)	100	0.1/0.1/0.2	0.1	0.2
HRNN All (*large*)	50	0.0/0.2/0.2	0.05	0.3
HRNN Init (*small*)	50	0.0/0.1/0.0	0.1	0.0
HRNN Init (*large*)	100	0.1/0.2/0.2	0.1	0.1

Table 5.2: *Network hyper-parameters on XING († RNN and RNN Concat use the same values).*

Model	Batch size	Dropout	Learning rate	Momentum
RNN (*small*)†	50	0.2	0.05	0.3
RNN (*large*)†	100	0.2	0.1	0.3
HRNN All (*small*)	50	0.5/0.4/0.4	0.05	0.5
HRNN All (*large*)	50	0.1/0.1/0.3	0.05	0.3
HRNN Init (*small*)	50	0.1/0.5/0.5	0.1	0.0
HRNN Init (*large*)	50	0.0/0.5/0.2	0.05	0.0

Table 5.3: *Network hyper-parameters on VIDEO († RNN and RNN Concat use the same values).*

Neural models were trained on Nvidia K80 GPUs equipped with 12GB of GPU memory[3]. Training times vary from $\sim$5 minutes for the small RNN model on XING to $\sim$30 minutes for the large *HRNN All* on VIDEO. Evaluation took no longer than 2 minutes for all the experiments. We want to highlight that training times do not significantly differ between RNN and HRNNs, with *HRNN All* being the most computationally expensive model due to the higher complexity of its architecture (see Figure 5.1).

5.3.3 Results

We evaluate with respect to the sequential next-item prediction task described in Section 2.6.1, i.e. given an event of the user session, we evaluate how well the algorithm predicts the following event. All RNN-based models are fed with events in the session one after the other, and we check the rank of the item selected by the user in the next event. In addition, HRNN models and RNN Concat are *bootstrapped* with all the sessions in the user history that precede the testing one in their original order. This step slows down the evaluation but it is necessary to properly set the internal repre-

[3]We employed Amazon EC2 *p2.xlarge* spot instances in our experiments.

		XING			VIDEO		
		Recall@5	**MRR@5**	**Precision@5**	**Recall@5**	**MRR@5**	**Precision@5**
	Item-KNN	0.0697	0.0406	0.0139	0.4192	0.2916	0.0838
	PPOP	0.1326	**0.0939**	0.0265	0.3887	0.3031	0.0777
small	RNN	0.1292	0.0799	0.0258	0.4639	0.3366	0.0928
small	RNN Concat	*0.1358*	*0.0844*	*0.0272*	0.4682	0.3459	0.0936
small	*HRNN All*	*0.1334*†	*0.0842*	*0.0267*†	*0.5272*	*0.3663*	*0.1054*
small	*HRNN Init*	*0.1337*†	*0.0832*	*0.0267*†	*0.5421*	*0.4119*	*0.1084*
large	RNN	0.1317	0.0796	0.0263	0.5551	0.3886	0.1110
large	RNN Concat	*0.1467*	*0.0878*	*0.0293*	0.5582	*0.4333*	*0.1116*
large	*HRNN All*	***0.1482***†	*0.0925*	***0.0296***†	0.5191	0.3877	0.1038
large	*HRNN Init*	***0.1473***†	*0.0901*	***0.0295***†	***0.5947***	***0.4433***	***0.1189***

Table 5.4: *Results of Recall, MRR and Precision for $N = 5$ on the XING and VIDEO datasets. small networks have 100 hidden units for RNNs and 100+100 for HRNNs; large networks have 500 hidden units for RNNs and 500+500 for HRNNs). All the networks have statistically significant different (ssd.) results from the baselines (Wilcoxon signed-rank test $p < 0.01$). Networks ssd. from RNN are in italic; HRNNs ssd. from RNN Concat are underlined. Best values are in bold. All differences between HRNNs are significant excluded the values marked with † superscript.*

sentations of the personalized models (e.g., the user-level representation for HRNNs) before evaluation starts. Notice that the evaluation metrics are still computed only over events in the test set, so evaluation remains fair. In addition, we discarded the first prediction computed by the RNN Concat baseline in each test session, since it is the only method capable of recommending the first event in the user sessions.

As recommender systems can suggest only few items at once, the relevant item should be amongst the first few items in the recommendation list. We therefore evaluate the recommendation quality in terms of Recall@5, Precision@5 and Mean Reciprocal Rank (MRR@5). In sequential next-item prediction, Recall@5 is equivalent to the hit-rate metric, and it measures the proportion of cases out of all test cases in which the relevant item is amongst the top-5 items. This is an accurate model for certain practical scenarios where no recommendation is highlighted and their absolute order does not matter, and strongly correlates with important KPIs such as CTR [60]. Precision@5 measures the fraction of correct recommendations in the top-5 positions of each recommendation list. MRR@5 is the reciprocal rank of the relevant item, where the reciprocal rank is manually set to zero if the rank is greater than 5. MRR takes the rank of the items into account, which is important in cases where the order of recommendations matters.

Table 5.4 summarizes the results for both XING and VIDEO datasets.

We trained each neural model for 10 times with different random seeds [4] and report the average results. We used Wilcoxon signed-rank test to assess significance of the difference between the proposed HRNN models and the state-of-the-art session-based RNN and the naïve personalization strategy used by RNN Concat.

Results on XING On this dataset, the simple personalized popularity baseline is a very competitive method, capable of outperforming the more sophisticated Item-KNN baseline by large margins. As prior studies on the dataset have already shown, users' activity within and across sessions has a high degree of repetitiveness. This makes the generation of 'non-trivial' personalized recommendations in this scenario very challenging [1].

This is further highlighted by the poor performance of session-based RNN that is always significantly worse than PPOP independently of the capacity of the network. Nevertheless, personalized session-based recommendation can overcome its limitations and achieve superior performance in terms of Recall and Precision with both small and large networks. HRNNs significantly outperform RNN Concat in terms of Recall and Precision (up to +3%/+1% with small/large networks), and provide significantly better MRR with large networks (up to +5.4% with *HRNN All*). Moreover, HRNNs significantly outperform the strong PPOP baseline of $\sim 11\%$ in Recall and Precision, while obtaining comparable MRR. This is a significant result in a domain where more trivial personalization strategies are so effective.

The comparison between the two HRNN variants does not highlight significant differences, excluded a small ($\sim 2\%$) advantage of *HRNN All* over *HRNN Init* in MRR. The differences in terms of Recall and Precision are not statistically significant. Having established the superiority of HRNNs over session-based recommendation and trivial concatenation, we resort to the VIDEO dataset to shed further light on the differences between the proposed personalized session-based recommendation solutions.

Results on VIDEO The experiments on this dataset exhibit drastically different results from the XING dataset. Item-KNN baseline significantly outperforms PPOP, and session-based RNN can outperform both baselines by large margins. This is in line with past results over similar datasets [61,62].

[4] The random seed controls the initialization of the parameters of the network, which in turn can lead substantially different results in absolute terms. Even though, we have not observed substantial differences in the relative performances of neural models when using different random seeds.

History length	XING	VIDEO
Short (≤ 6)	67.00%	53.69%
Long (> 6)	33.00%	46.31%

Table 5.5: *Percentage of sessions in each history length group.*

RNN Concat has comparable Recall and Precision with respect to session-based RNNs and interestingly significantly better MRR when large networks are used. This suggest that straight concatenation does not enhance the retrieval capabilities of the RNN recommender but strengthens its ability in ranking items correctly.

However, *HRNN Init* has significantly better performance than all baselines. It significantly outperforms all baselines and RNNs (up to 6.5% better Recall and 2.3% MRR wrt. RNN Concat with large networks). In other words, also in this scenario the more complex cross-session dynamics modeled by HRNN provide significant advantages in the overall recommendation quality. We will investigate the possible reasons for these results in the following sections. It is worth noting that *HRNN All* performs poorly in this scenario. We impute the context-enforcing policy used in this setting for the severe degradation of the recommendation quality. One possible explanation could be that the consumption of multimedia content (videos in our case) is a strongly session-based scenario, much stronger than the job search scenario represented in XING. Users may follow general community trends and have long-term interests that are captured by the user-level GRU. However, the user activity within a session can be totally disconnected from her more recent sessions, and even from her general interests (for example, users having a strong general interest over extreme-sport videos may occasionally watch cartoon movie trailers). *HRNN Init* models the user taste dynamics and lets the session-level GRU free to exploit them according to the actual evolution of the user interests within session. Its greater flexibility leads to superior recommendation quality.

5.3.4 Analysis on the user history length

We investigate deeper the behavior of Hierarchical RNN models. Since we expect the length of the user history to have an impact on the recommendation quality, we breakdown the evaluation by the number of sessions in the history of the user. This serves as a proxy for the 'freshness' of the user within the system, and allows to evaluate recommenders under different amounts of historical information about the user modeled by the user-level

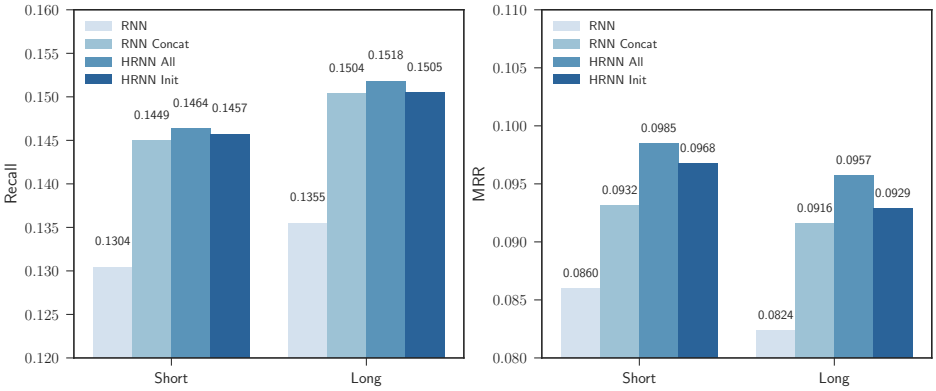


Figure 5.3: *Recall@5 and MRR@5 on XING grouped by user history length.*

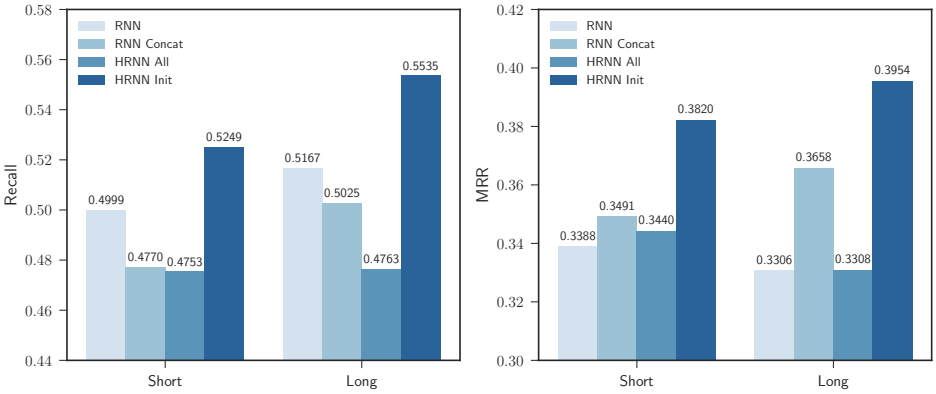


Figure 5.4: *Recall@5 and MRR@5 on VIDEO grouped by user history length.*

GRU before a session begins. To this purpose, we partitioned user histories into two groups: 'Short' user histories having ≤ 6 sessions and 'Long' user histories having 6 or more. The statistics on the fraction of sessions belonging to each group for both datasets are reported in Table 5.5. Since our goal is to measure the impact of the complex cross-session dynamics used in HRNN wrt. traditional RNN, we restrict these analysis to RNN-based recommenders in the *large* configuration. For each algorithm, we compute the average Recall@5 and MRR@5 per test session grouped by history length. The analysis on Precision@5 returns similar results to Recall@5, so we omit it here also for space reasons. To enhance the robustness of the experimental results, we run the evaluation 10 times with different random seeds and report the median value per algorithm.

Figure 5.3 shows the results on XING. As the length of the user his-

tory grows, we can notice that Recall slightly increases and MRR slightly decreases in MRR for all methods, session-based RNN included. The relative performance between methods does not changes significantly between short and long user histories, with *HRNN All* being the best performing model with 12% better Recall@5 and 14-16% better MRR@5 wrt. session-based RNN. *HRNN Init* has performance comparable to RNN Concat and *HRNN All* in accordance to our previous findings.

Figure 5.4 shows the results on VIDEO. Recall of all methods – session-based RNN included – improves with the length of the user history. MRR instead improves with history length only for RNN Concat and *HRNN Init*. This highlights the need for effective personalization strategies to obtain superior recommendation quality at session-level for users that heavily utilize the system. Moreover, the performance gain of *HRNN Init* wrt. session-based RNN grows from 5%/12% (short) to 7%/19% (long) in Recall@5/MRR@5, further highlighting the quality of our personalization strategy. Coherently with our previous findings, *HRNN All* does not perform well in this scenario, and its performances are steady (or even decrease) between the two groups.

In summary, the length of the user history has a significant impact on the recommendation quality as expected. In loosely session-bounded domains like XING, in which the user activity is highly repetitive and less diverse across sessions, enforcing the user representation in input at session-level provides slightly better performance over the simpler initialization-only approach. However, in the more strongly session-based scenario, in which the user activity across sessions has a higher degree of variability and may significantly diverge from the user historical interest and tastes, the simpler and more efficient *HRNN Init* variant has significantly better recommendation quality.

5.3.5 Analysis within sessions

Here we breakdown by number of events within the session in order to measure the impact of personalization within the user session. We limit the analysis to sessions having length ≥ 5 (6,736 sessions for XING and 8,254 for VIDEO). We compute the average values of each metric groped by position within the session (Beginning, Middle and End). Beginning refers to the first 2 events of the session, Middle to the 3rd and 4th, and End to any event after the 4th. As in the previous analysis, we focus on RNN-based models in the *large* configuration and report the median of the averages values of each metric computed over 10 runs with different random seeds.

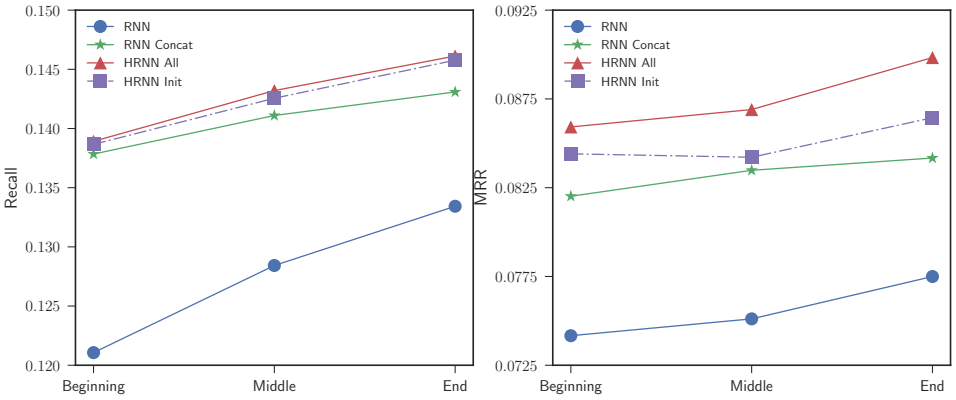


Figure 5.5: *Recall@5 and MRR@5 on XING for different positions within session.*

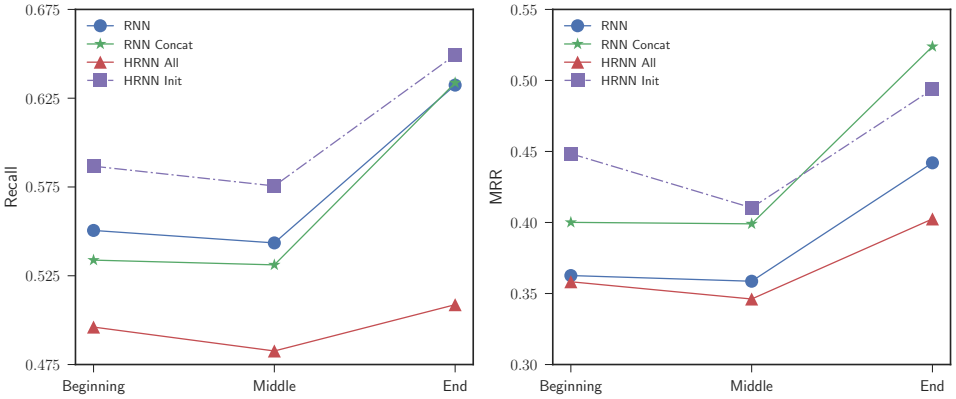


Figure 5.6: *Recall@5 and MRR@5 on VIDEO for different positions within session.*

Results for Recall@5 and MRR@5 are shown in Figure 5.5 and Figure 5.6 for XING and VIDEO respectively.

On XING, the performance of all methods increses with the number of previous items in the session, suggesting that the user context at session-level is properly leveraged by all RNN-based models. However, there is a wide margin between personalized and 'pure' session-based models. Both HRNNs have similar Recall@5 and are comparable to RNN Concat. Interestingly, the gain in MRR@5 of *HRNN All* wrt. to both RNN and RNN Concat grows with the number of items processed, meaning that in this scenario the historical user information becomes more useful as the user session continues. *HRNN Init* has constantly better MRR that RNN Concat, with wider margins at the beginning and at the and of the session.

On VIDEO, the behavior within session is different. We can notice that both Recall and MRR increase between the beginning and end of a session as expected. *HRNN Init* exhibits a large improvement over RNN and RNN Concat at the beginning of the session (up to 10% better Recall and 25% better MRR). This conforms with the intuition that past user activity can be effectively used to predict the first actions of the user in the forthcoming session with greater accuracy. After the first few events the gain in Recall of personalized over pure session-based models reduces, while the gain in MRR stays stable. In other words, after a few events, the session-level dynamics start to prevail over longer-term user interest dynamics, making personalization strategies less effective. However, personalization still provides superior ranking quality all over the session, as testified by the higher MRR of both *HRNN Init* and RNN Concat over RNN. It is important to notice that better recommendations at beginning of a session are more impactful than later in the session because they are more likely to increase the chances of retaining the user. Finally, *HRNN All* is always the worse method, further underpinning the superiority of the *HRNN Init* variant.

5.3.6 Experiments on a large-scale dataset

We validated HRNNs over a larger version of the VIDEO dataset used in the previous experiments. This dataset is composed by the interactions of 810k users on 380k videos over the same 2 months periods, for a total of 33M events and 8.5M sessions. We then applied the same preprocessing steps used for the VIDEO dataset. We named this dataset VIDEOXXL. Due to our limited computational resources, we could only test small networks (100 hidden units for RNN and for 100+100 HRNNs) on this large-scale dataset. We run all RNNs and HRNNs once using the same hyper-parameters learned on the small VIDEO dataset. Although not optimal, this approach provides a first approximation on the applicability of our solution under a more general setting. For the same reason, we do not provide an exhaustive analysis on the experimental results as done for the smaller datasets. To speed up evaluation, we computed the rank of the relevant item compared to the 50,000 most supported items, as done in [62].

Results are summarized in Table 5.6 and confirm our previous findings on the small VIDEO dataset. RNN Concat is not effective and performs similarly to session-based RNN. On the other hand, Hierarchical RNNs outperform session-based RNN by healty margins. In particular, *HRNN Init* outperforms session-based RNN by $\sim$28% in Recall@5 and by $\sim$41% in MRR@5. These results further confirm the effectiveness of the HRNN

	VIDEOXXL		
	Recall@5	**MRR@5**	**Precision@5**
RNN	0.3415	0.2314	0.0683
RNN Concat	0.3459	0.2368	0.0692
HRNN All	0.3621	0.2658	0.0724
HRNN Init	**0.4362**	**0.3261**	**0.0872**

Table 5.6: *Results on the VIDEOXXL dataset for small networks. Best values are in bold.*

models presented in this paper, and further underpin the superiority of *HRNN Init* over the alternative approaches for personalized session-based recommendation.

5.4 Conclusions

In this chapter we addressed the problem of personalizing session-based recommendation by proposing a model based Hierarchical RNN, that extends previous RNN-based session modeling with one additional GRU level that models the user activity across sessions and the evolution of her interests over time. HRNNs provide a seamless way of transferring the knowledge acquired on the long-term dynamics of the user interest to session-level, and hence to provide personalized session-based recommendations to returning users. The proposed HRNNs model significantly outperform both state-of-the-art session-based RNNs and the other basic personalization strategies for session-based recommendation on two real-world datasets having different nature. In particular, we noticed that the simpler approach that only initializes the session-level representation with the evolving representation of the user (*HRNN Init*) gives the best results. We delved into the dynamics of session-based RNN models within and across-sessions, providing extensive evidences of the superiority of the proposed *HRNN Init* approach and setting new state-of-the-art performances for session-based recommendation.

There are several opportunities for extension. For example, it can be interesting to investigate over other domains like personalized music recommendation, e-commerce and advertisement. Specifically for e-commerce, previous studies has shown the importance of the long-term and short-term dynamics of the user interest on the recommendation quality in session-based scenario [66]. Unfortunately, there do not have access to any dataset suitable for this task at the moment of writing of this thesis. We are therefore forced to leave this aspect to future works.

Another interesting aspect that is certainly worth investigating is the how

item and user features can be effectively added to the HRNN model. As we have already seen in Chapter 4, item features can be effectively used to improve the ranking quality in session-based recommendation. Since item features can highlight certain properties of items that are of special interest for the user, by including the into our HRNN model we could be able to refine the user representation over time (in combination with user metadata when available) and improve session-based recommendation even further.

Part IV

Sequence-Aware Recommendation in Music

CHAPTER 6

Modeling Musical Taste Evolution with Recurrent Neural Networks

Musical preference dynamics present a rich and nebulous problem space for the development of models capable of predicting the temporal structure of user behavior. Music listening is an extremely long-tailed distribution, where a massive proportion of user behavior is dominated by an industry that's constantly delivering new pop stars in addition to the ebb and flow of popularity among mega-artists with careers spanning multiple decades.

However, outside of this "short head" behavior, there exists a majority of listeners who have interest spanning over hundreds of thousands of artists and millions of tracks. Their music discovery is influenced by a multitude of factors such as social, geographical, generational, or musicological. They may be going through a maturation of preference in a discovery of 1950's bop jazz or perhaps developing a guilty pleasure in 1990's pop music.

Despite the richness and complexity of this domain, the evolution of music preference over time has received very little attention in the recommendation literature to date. At the time of writing, we are only aware of one paper that has addressed the topic [97].

In this chapter, we set out to provide a deep dive into this topic and to develop models which are capable of encoding the evolution of listener music preferences. Our ultimate goal is to provide extremely powerful recommendations, but this must be achieved through the ability to predict where their preference is headed next.

To construct our dataset of musical preferences, we leverage internet radio data from Pandora[1], a US-based music streaming service with over 80 million monthly active listeners. We compile a dataset of personalized radio station creation, sampling from a dataset of over 11 billion stations that have been created to date. These stations can be created (or "seeded") from an artist, track, composer or genre, and become personalized to the listeners tastes as they provide feedback. Listeners add stations to their profile strictly in a sequential manner over time, in synchrony with the evolution of their tastes. Modeling station creation over time allows us to look at the dependency structure over these potentially large changes in listening behavior.

It is worth stressing out that station creation represents the interest of the user not only on the single artist, genre or track, that serves as seed for the station, but on the *whole collection of songs* that stems from it. It is therefore a much stronger (and rare) feedback than likes/dislikes on individual artists/genres/tracks. For these reasons, we believe the flow of stations created by the user to be a good proxy of the evolution of her musical tastes over time.

The work presented in this chapter was run in collaboration with a group of researchers from *Pandora* [2]. The outline of this chapter is the following. In Section 6.1 we present the background and related works. In Section 6.2 we present sequence-aware models used in our experiments. In Section 6.3 we present the experiments and discuss about the results. In Section 6.5 we present the conclusions and the possible future developments.

6.1 Background and Related works

Modeling user preference dynamics over time has been the subject of investigation across multiple recommendation domains [73, 97]. In recommender systems, modeling the evolution of user tastes over time is crucially important in order to to react to drift in the user's preferences and to adapt recommendations accordingly.

[1] `www.pandora.com`

[2] The author spent the summer 2016 at *Pandora Media, Inc.* as a research intern.

Time-aware recommender systems (TARS) [21] exploit time as an additional contextual dimension that is added to traditional recommender systems. In these systems, ratings timestamps are exploited to identify periodicity in user habits. In general, the goal is to improve the accuracy of collaborative filtering models like matrix factorization by adding temporal dynamics to otherwise static profiles [22, 73]. A downside to this family of models is that they are dependent on the exact timestamps at which user feedback is collected, which in turn may not correspond to the actual consumption time. Moreover, users who have a similar evolution in their tastes over different temporal scales will be treated differently by these models.

Sequence-aware recommender systems (SARS), the subject of this thesis, provide the necessary flexibility by relaxing the aforementioned temporal constrains and focusing exclusively on the sequential order of the historical user activity.

In the music domain, sequential features extracted from historical listening records can be used to generate coherent continuations to the current listening session or playlists [18, 67]. To our knowledge, only [97] considered a sequence based approach to modeling the evolution of user tastes through time in music. User and song transitions are embedded into a common latent first-order Markov space that can browsed with Gaussian random walks. However, this approach is affected by the same limitation in the size of the state space of Markov models, which in turn severely constraints the number of users and songs that can be effectively represented.

This work is the first one that uses Recurrent Neural Networks for modeling user preference dynamics in music to our knowledge.

6.2 Models

In this section we describe our model (and other methods) to capture the musical taste evolution of the listeners and in particular to solve the problem of listener's next searched-created stations (next-station in short hereafter).

6.2.1 Recurrent Neural Network

Recurrent Neural Networks have been thoroughly examined in Chapter 4 and Chapter 5. We recall their formulation here to the sake of readability. The Recurrent Neural Network (RNN) computes for each station s_t a dense vector h_t, the recurrent state, that combines the previous s_t with the previous state h_{t-1} according to the formula,

$$h_t = f(s_t, h_{t-1}), h_0 = 0, \tag{6.1}$$

where $h_t \in \mathbb{R}^{d_t}$, where d_t is the number of dimensions of the recurrent state, and f is a non-linear transformation. In this work we adopted the Gated Recurrent Unit (GRU) model for the modeling of click-stream data described in [62]. Each station s_n is represented as one-hot vector, i.e., a vector of length equal to the number of stations S with a 1 corresponding to the index of the current station, and 0 otherwise. The parametrization f of GRU is given by:

$$\begin{aligned} r_t &= \sigma\left(U_r s_t + W_r h_{t-1}\right), && \textit{(reset gate)} \\ z_t &= \sigma\left(U_z s_t + W_z h_{t-1}\right), && \textit{(update gate)} \\ \tilde{h}_t &= \tanh\left(U_h z_t + W_h (r_t \cdot h_{t-1})\right), && \textit{(candidate update)} \\ h_t &= (1 - z_t) h_{t-1} + z_t \tilde{h}_t, && \textit{(final update)} \end{aligned} \tag{6.2}$$

where σ is the logistic sigmoid, $\cdot$ is the element-wise scalar product between vectors, $U_r, U_z, U_h \in \mathbb{R}^{d_h \times S}$ and $W_r, W_z, W_h \in \mathbb{R}^{d_h \times d_h}$. The GRU predicts the next station in the sequence according the following,

$$\hat{s}_{t+1} = g(h_{t-1}), \tag{6.3}$$

where g is another non-linear transformation that projects the recurrent representation of the GRU onto the space of the stations. The formulation of g depends on the loss function used to train the GRU. In accordance to [62], we initially experimented with three different loss functions: cross-entropy, Bayesian Pairwise Ranking (BPR) and TOP-1 loss. With cross-entropy $\hat{s}_{t+1} = \mathrm{softmax}(W_y h_{t-1})$, whereas with BPR and TOP-1 $\hat{s}_{t+1} = \tanh(W_y h_{t-1})$. We settled on BPR loss function since it yields the best results. A graphical description of this model is provided in Figure 6.1.

The GRU is trained to predict the next station given the complete past history of the user that is encoded into its recurrent state h_t. The training is performed end-to-end such that sequential relationships between stations are learned directly from data without the need of ad-hoc feature extraction.

In conformity with the previous chapters, we used a single layer GRU network trained with parallel mini-batches and negative output sampling in our experiments. These features are of crucial importance for training the model over the large dataset used for our experiments.

6.2.2 Similarity-based baseline models

In addition to the RNN, we have considered several similarity-based methods for next-station prediction. These methods typically assign a score to each candidate next-station that is proportional to its similarity with the previous station in the sequence. In this work, we have defined item similarity

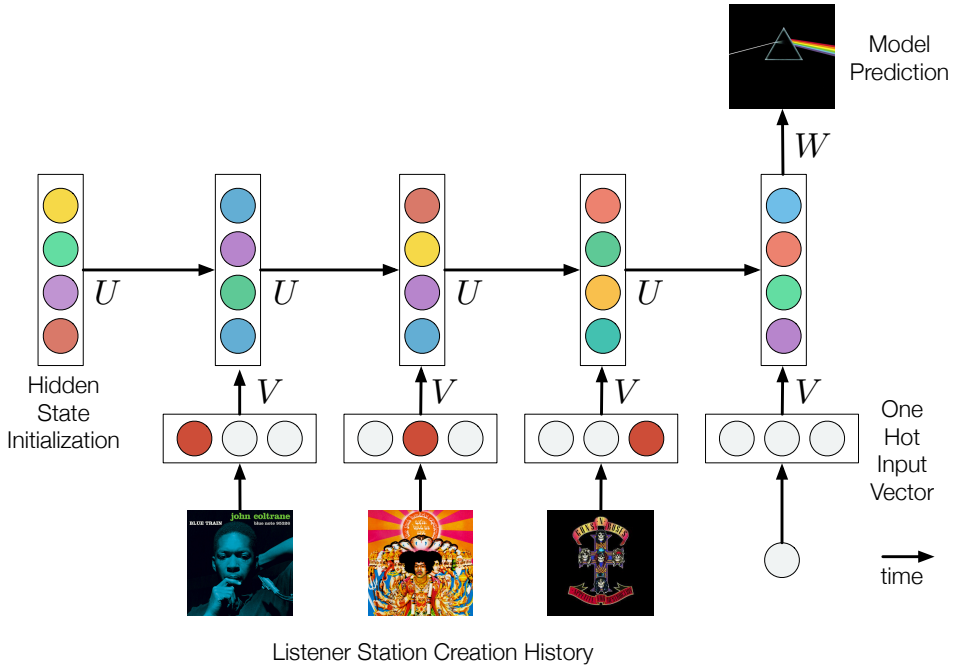


Figure 6.1: *RNN model at a high level. The red dots in the input layer represent the one-hot encoding of the input and the colors in the hidden layer represent the model activations.*

wrt. co-occurrence of stations in user profiles. Specifically, we have considered the overall item popularity, co-occurrence [61] and Word2Vec [53] based similarities. Despite their apparent simplicity, these models turn out to be very strong baselines in this domain, as we will show in the experimental evaluation.

Item Popularity (*POP*)

Given the extremely long tailed distribution of music, it's extremely important to consider a popularity baseline for any recommendation approach. For this baseline, we simply take the most popular station seeds (i.e., artist, genre, composer, track) across the service and utilize the top K as the recommendations.

Co-occurrence kNN (*KNN*)

A simple way of assessing the similarity between items is to consider how frequently they co-occur. More concretely, we compute how many times two different items occur in the same sequence of user interactions. This

rather simple approach, that leads to recommendations of the type "others who added this station also added these other ones," has proven its effectiveness in a variety of domains, most notably in e-commerce [79].

In this work, we have to the co-occurrence item-item similarity metric used in [61] for next-click recommendation in click-stream data. The similarity between two items i and j is computed as,

$$\text{sim}(i,j) = \frac{|S(i) \cap S(j)|}{\sqrt{|S(i)| + \lambda} \cdot \sqrt{|S(j)| + \lambda}}, \tag{6.4}$$

where $S(z)$ returns the sessions (sequences) in the training database where the item z occurs at least once, and λ is a damping factor to avoid coincidental high similarities of rarely visited items. At recommendation time T, all items are ranked according to their similarity w.r.t. the last item in the sequence i, and then the top-k similar items to the last one in the sequence are recommended to the user.

The damping factor is utilized in order to mitigate the impact of data sparsity on the similarity metric, which is otherwise strongly constrained by the actual co-occurrences that are observed in the data. Despite these challenges, this approach showed very strong performance in our experiments.

Word2Vec kNN (*W2V*)

In addition to item co-occurrence, more complex interactions between items can be modeled from the sequences of user interactions. In particular, we adapt *prod2vec* [53] to the station recommendation case. The *prod2vec* model involves learning vector representations of products from sequential actions of users. In our case, we learn a vector-wise representation of stations from the sequence of station created by users. In particular, we consider the sequence as "sentence" and single station as "words." Further details on this method can be found in [53].

6.3 Experiments

In the following experiments we will investigate employing a variety of models for predicting listener preference dynamics. After an explanation of the dataset and metrics, we will evaluate our model in the task of predicting the next stations that a listener will create.

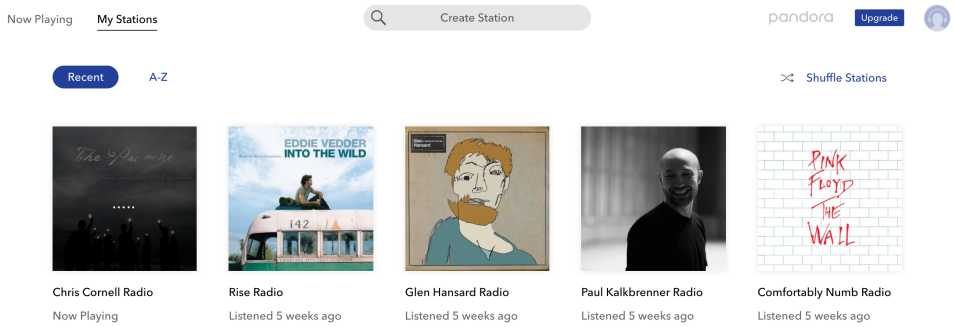


Figure 6.2: *Example of the recently played radio-stations. To model the evolution of the user's musical tastes, we considered the sequence of stations created only from the search-box (at the top of the picture) to filter the biases due to the recommendations offered in other parts of the application. This screen-shot was captured from the Pandora web-app.*

Number of Listeners	330,170
Number of Stations	3,079,399
Number of Listeners (Training)	324,750
Number of Stations (Training)	2,912,564
Number of Listeners (Test)	64,750
Number of Stations (Test)	151,015

Table 6.1: *Statistics of the dataset.*

6.3.1 Dataset

For our experiments, we have assembled a data collection containing the stations added by a random sample Pandora listeners over a 1-year period. We consider the stations together with the time-stamp in which they were created. To reduce the impact of selection bias [122] induced by the existing recommendation systems used by the platform to suggest new stations to listeners, we considered only stations added through the search-box (see Figure 6.2). While it is still possible that listener choices are influenced by recommendations received in other parts of the application (e.g., bias present in the process of track selection on an existing station exposing a listener to a new artist), searches through the search-box are clear expressions of the intent of the listener of looking for a new station. These are less-biased signals that we believe are good indicator of listener interests and hence adequate for modeling the actual interests of the listener and their evolution through time.

Details about our dataset can be found in Table 6.1. We collected se-

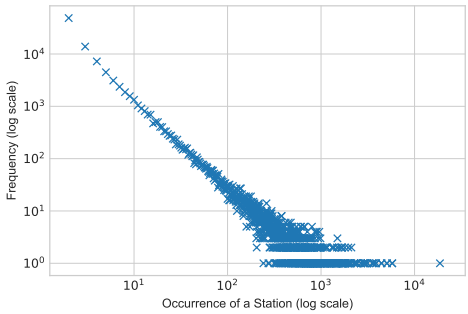


Figure 6.3: *The distribution of stations by popularity in log-log scale. The occurrence of user created stations represents popularity of a station, in x-axis. The number of such stations is shown in the y-axis. Very popular stations such as Today's Hits Radio has a high occurrence in creation, but there exists only a few of these stations.*

quences of stations added by ~300k listeners over a 1-year period. Figure 6.3 depicts the distribution of station creation. As previously discussed in the introduction of this chapter, music popularity follows a power-law distribution where a very small number of stations receive an overwhelming share of total creation.

To create training and testing datasets, we partitioned our dataset temporally, considering the first 11 months for training and the last month for testing.

As our ultimate focus is to predict how a listener's taste will evolve, we focus exclusively on listeners who have created a minimum of three stations on the service. Listeners who have created only one or two stations are filtered from our dataset. Furthermore, to remove spurious stations from our training dataset we also removed any station which was listened for less than one hour. This leads to a slight overall reduction of the number of stations both through this filtering as well as our restriction to include only stations created via listener searches. The final training/testing dataset sizes are reported in Table 6.1.

6.3.2 Experimental Setup

Given a set of stations created by the listener in the training set u_{train} = (s_1, s_2, s_3, ..., s_t) and a set of stations created by the same listener in the test set u_{test} = (s_{t+1}, ..., s_n), in order to evaluate our algorithms we aim to predict the test stations u_{test}. As usual, we evaluate against the sequential next-station prediction task (Section 2.6.1) and we use standard information

	Recall			
	@5	@10	@20	@50
200	0.0264	0.0441	0.0708	0.1273
500	0.0275	0.0460	0.0748	0.1367
1000	**0.0282**	**0.0483**	**0.0781**	**0.1378**

Table 6.2: *Recall@K for different hidden unit sizes.*

retrieval measures as evaluation metrics. For each listener we compute:

- *MRR@K*: Mean reciprocal rank is the inverse of the position of the first correct station in the ranking produced by the algorithm up to position K.
- *Recall@K*: The fraction of stations correctly predicted in K suggestions divided by overall number of stations created by the listener in the test set.

RNN Parameter Tuning

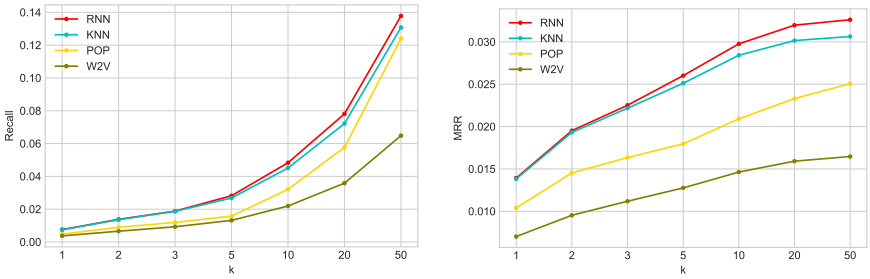
Our models are implemented in python based on the package Theano[3] 0.9.0, and experiments are performed on NVIDIA Tesla M40 GPUs. As mentioned in Section 6.2, we used Bayesian Pairwise Ranking as the loss function. For each size of the hidden units we tuned a set of parameters using random search. The tuning was performed on a validation set that is of the same nature as the test set, i.e. contains a set of listeners with an ordered sequences of stations created. We have tuned the networks for Recall@10. We found the optimal learning rate to be 0.05 for all the unit sizes. We set the momentum to 0.1, and dropout to 0.1.

In order to understand the impact of the hidden unit size on effectiveness, we trained initially the network with 3 different unit size: 200, 500, 1000. Table 6.2 reports Recall@K metrics for different sizes of hidden units. Recall increases with more units for all the K. However, units larger than 1000 led to computational cost limitations without significant performance gains. We choose 1000 units for the rest of the experiments.

6.4 Results

We report general results of next-stations prediction, in Recall in Figure 6.4a, and MRR in Figure 6.4b, for different prediction cutoff K ranging

[3] http://deeplearning.net/software/theano/



(a) *Recall@K RNN and baseline algorithms.* **(b)** *MRR@K RNN and baseline algorithms.*

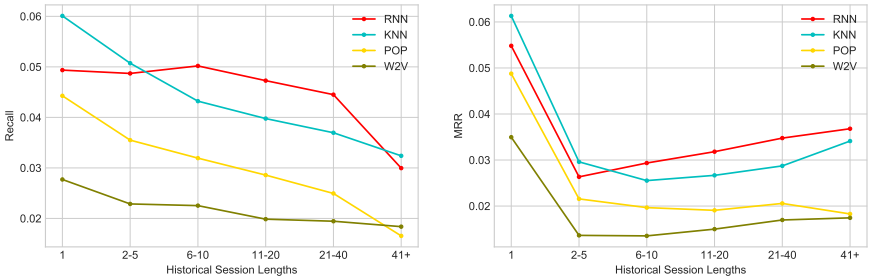
	@1	@5	@10	@20	@50
Recall					
KNN	0.0074	0.0268	0.0451	0.0722	0.1307
RNN	0.0075	0.0282	0.0483	0.0781	0.1378
Δ %	+1.9%	+5.1%	+7.1%	+8.2%	+5.4%
MRR					
KNN	0.0138	0.0251	0.0284	0.0302	0.0306
RNN	0.0139	0.0260	0.0298	0.0320	0.0326
Δ %	+0.8%	+3.6%	+4.7%	+6.0%	+6.5%

Table 6.3: *Overall Recall@K and MRR@K for RNN and the best baseline KNN.*

from 1 to 50. In all values of K, *KNN* is the best performing baseline model, outperforming popularity (*POP*), and word2vec (*W2V*) by a large margin.

RNN outperforms the best baseline model *KNN* when $K > 10$, with +8.20% in Recall@20, and +5.41% in Recall@50. For small K (≤ 10), there is no big advantage in recall. However, in MRR metric, even at small $K = 5$, and $K = 5$, RNN ranks better than *KNN* by 1.57% and 3.62%, respectively. In Figure 6.4b, we can see that with increasing K value, RNN's margin of improvement over *KNN* also increases consistently in MRR. Detailed result comparisons of RNN and *KNN* are reported in Table 6.3.

While we expected the RNN model to predict next stations better because it specifically modeled sequential behavior of listener's taste, we want to dive deep in listener analysis to substantiate this claim. Typically this is done through slicing and dicing based on listener characteristics.



(a) *Recall@10 RNN and baseline algorithms, segmented by historical session length.*

(b) *MRR@10 RNN and baseline algorithms, segmented by historical session length.*

6.4.1 Listener Segmentation Study

We perform several listener segment studies. In this study we fix $K = 10$ for both Recall and MRR metrics. First, we segment listeners based on their historical session length (number of stations created by a listener in the training set), represented by the number of previously created stations before the test period. We further bin listeners into 1, 2-5, 6-10, 11-20, 21-40, and 41+ previous stations. The fewer stations a listener has, the less listening history data we have. Figure 6.5a and Figure 6.5b show that both RNN and *KNN* outperforms *POP* and *W2V* in all listener segments in Recall@10 and MRR@10. For listeners with very short history, i.e., ≤ 5 stations, *KNN* is the best algorithm. However, for listeners with more than 5 stations, i.e. when sequential patterns might be emerging, RNN algorithm overtakes *KNN* by 11.9% in Recall@10. We attribute this to neural network's ability to retain memory. This suggests that our 1 layer network structure is very efficient at learning preference changes when given enough history. Yet it is not wise to use it as a standalone model as *KNN* clearly claims an advantage with listeners with a very short history. More in-depth study of ensembles is given in Section 6.4.3. Note that we included metrics for listeners who created more than 41+ stations for completeness, even though very few such listeners exists in the dataset.

Armed with the powerful insight that RNN algorithms perform much better for a subset of listeners, we focus on this set of listeners with > 5 previous stations at prediction time. In Table 6.4, we report the Recall@K and MRR@K results for this filtered set of listeners. RNN exhibits large performance gain in both Recall and MRR metrics in all K values over best baseline *KNN*. We compare performance by segmenting further on the listeners by next session lengths, and show results in Figure 6.6. The high-

	Recall				
	@1	@5	@10	@20	@50
KNN	0.0073	0.0266	0.0438	0.0705	0.1302
RNN	0.0081	0.0290	0.0490	0.0808	0.1436
Δ %	+11.5%	+8.9%	+11.9%	+14.6%	+10.2%
	MRR				
KNN	0.0134	0.0245	0.0278	0.0298	0.0304
RNN	0.0149	0.0271	0.0310	0.0335	0.0338
Δ %	+10.9%	+10.2%	+11.7%	+12.5%	+11.2%

Table 6.4: *Recall@K and MRR@K for RNN and the best baseline KNN. Training restricted to listener with more than 5 previous stations.*

light is that the RNN outperforms baselines by a larger and larger margin as the listeners create more stations. This again suggests that RNN learns a sequential dependency that is not captured by other models.

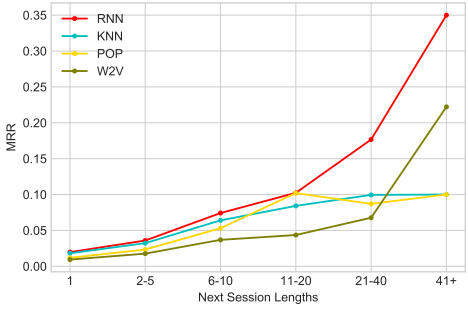


Figure 6.6: *MRR@10 RNN and baseline algorithms, segmented by next session length. RNNis more effective for listeners who create more stations in the next session.*

6.4.2 Long Tail Music

In personalized music services, the most difficult task is often to predict long tail (unpopular) music correctly. To further understand the difference among models in this regard, we single out the true positives from the test set for $K = 50$, i.e., the intersection of predicted stations and listener created stations, and report results segmented by station popularity. As mentioned in Section 6.3.1, the popularity of a station is represented by the number of occurrences in the training set. Here we bin the occurrence value with a bin size of 500.

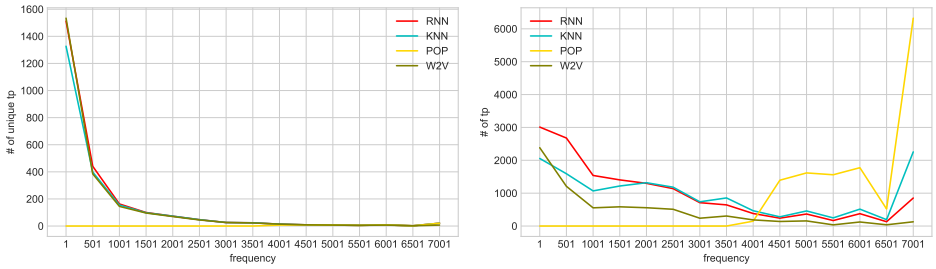


Figure 6.7: *Correctly recommended item (true positive) distribution comparison. In the left figure, RNN recommends the most amount of unique least popular items. In the right figure, RNN recommends 50% more least popular items than KNN in occurrence.*

Not surprisingly, Figure 6.7 showed the popularity algorithm predicts more very popular stations correctly. However, this is done at the expense of getting none of the long tail stations right, shown in right of Figure 6.7 as the yellow line at 0 true positives for stations with occurrence < 3500. Indeed, recall that in the general results in Figure 6.4a, *POP* model lagged behind both the RNN and the *KNN* models. It is unexpected and very encouraging that RNN in fact recommends almost 50% more least popular stations (those occurred less than 500 times) accurately than the best baseline, *KNN*, as shown in the right of Figure 6.7. In the left figure of Figure 6.7, we also show the RNN and *W2V* both predict the most unique number of true positives. This provides further evidence that as a listener's taste tends towards more nuanced and less popular, the recurrent neural network captures that tastes evolution more accurately than a simple nearest neighbor approach. Given that the RNN also outperforms all models in Recall@50 and MRR@50, we believe we demonstrated its value as a promising long tail music recommender.

6.4.3 Learning to Rank

We lastly look at a combination of our models in a cascaded late fusion ensemble in order to further investigate the predictive power of the RNN. Recall here that given the stations created by a listener in our training set, our goal is to predict the next stations a listener is going to create. For each listener, we generate the top-10 recommendation for each method and consider the union as the set of recommendation candidates to be re-ranked.

We consider the true target as the station created in the validation set by the same listener. We than label the true targets as relevant and all the others as non-relevant. As the final stage we use LambdaMART [141] as our

supervised ranker to re-rank the candidate list. We tune the LambdaMART model with 500 trees and the parameters are learned using standard separate training and validation set.

Features

For this learning to rank (L2R) approach we used a total of 11 features (a combination of contextual and suggestion features):

1. the inverse of rank of the candidate in *KNN*
2. the inverse of rank of the candidate in *POP*
3. the inverse of rank of the candidate in *W2V*
4. the popularity of the last station created
5. the typology of the last station created
6. the popularity of the candidate station
7. the typology of the candidate station
8. the age of the listener
9. how diverse are the songs listened by the listener
10. how popular are the song listened by the listener
11. the number of stations created by the listener in the training set

+RNN

The proposed RNNcontributes to one additional feature to the previous mentioned one. In this case we include the top-10 recommendation generated by RNN to the set of candidates to be re-ranked. We optimize based on MRR@1, MRR@5.

L2R Results

Table 6.5 reports the improvement in terms of MRR obtained by using RNN as additional feature in our learning to rank system previously described[4]. The rows represents two different ways we optimized the system: on metric MRR@1 and metric MRR@5 (note that this is done in the training phase). We then compute the MRR metric on test and report the improvement in percentage terms, particularly MRR@1, MRR@3 and MRR@5. We aim to improve the top ranking results, hence the choice of small K values. Results show an improvement of over 16% in MRR@5 when integrating RNN as additional feature in our learning to rank system. In general, there is a large improvement in all cases presented.

[4]The results on the full dataset show a similar behavior.

	@1	@3	@5
+RNN (opt. MRR@1)	+10.3%	+13.8%	+16.7%
+RNN (opt. MRR@5)	+9.4%	+13.5%	+16.1%

Table 6.5: *Learning to rank results. Increase in terms of MRR obtained by using a learning to rank +RNN. We optimize the learning to rank on MRR@1, MRR@5, MRR@10. Results reported are for listeners with historical session length > 5.*

Together with observations from Section 6.4.2, we interpret this results as a substantial amount of the predictions and the ranking of predictions obtained by RNN are *complimentary to other models*.

6.5 Conclusions

This work has presented an investigation into a novel domain in the space of modeling listener music preference dynamics. We showed the RNN model to be extremely effective in modeling the evolution of these preferences, offering a +8.2% increase in Recall@20 over other baselines.

We have found these performance advantages become more pronounced as more historical data is available for a given user. The RNN outperforms the other baselines most substantially when a listener has between 6 and 40 stations. This result indicates that long-term dependency structure is important and that the true advantage of the RNN is its ability to encapsulate it. With less than 6 stations it is perhaps the case that the data is too limited to learn this dependency structure. The decrease above 40 stations is likely related to a small number of listeners having such high station counts.

One of the most exciting advantages of the RNN is its ability to correctly recommend less popular items. Items that appeared the least were recommended correctly by the RNN than any other model.

For an additional assessment of the predictive power of the RNN, we also investigated a cascaded learn-to-rank model which allows us to analyze the information gain delivered by the RNN over the other baselines. We showed a 16% increase in total performance in MRR@5.

In future work we plan to investigate deeper architectures as well stronger ways to integrate side information. Architectures that incorporate features like genre and popularity about the station as well as demographics about the listener seem like they could be particularly promising for this work. Another direction for investigation is to look at the problem in terms of listening behavior instead of station creation. These models would predict if someone is going to like a particular song.

CHAPTER 7

Automated Playlist Generation with Recurrent Neural Networks

Music is on of typical application domain of sequence-aware recommendation, as highlighted in our analysis of the state of the art in Chapter 2. Among the several recommendation tasks in music, one has received special attention from the the Music Information Retrieval and Recommender Systems communities, that of automated playlist generation and recommendation. According to the definition in [18], a *playlist* simply is a sequence of tracks. Playlist generation consists in the generation of a suitable sequence of songs given a pool of tracks, a background knowledge database and, optionally, some target characteristics of the playlist.

Despite the considerable amount of research available on the topic, little consensus was found on the importance of the order of songs in the playlist generation process. According to interviews with practitioners and postings to a dedicated playlist-sharing website, song order –or at least the relative position of songs within a playlist – has been identified as an important aspect when compiling a playlist [35]. Although some approaches to playlist generation take the song order into consideration, to the best of our knowledge, they do not explicitly analyze its importance, or the exist-

ing results are strongly dataset-dependent so that no single conclusion can be taken [18].

To shed some light on this issue, we studied the sequential generation of playlists, i.e. recommendations on the next song to add to the playlist are constantly updated according to the songs that are already available in the playlist. Consider that online radios, an already well established business sector with products like Pandora [1], can be seen as particular case of endless playlist generation, in which the songs are continuously recommended in a sequential fashion.

In this chapter, we model hand-curated music playlists with Recurrent Neural Networks and show their competitive performance on this task. We then perform numerical experiments in which we challenge the recurrent models by randomizing the order of songs within playlists. The experiments yield negligible performance differences, indicating that the choice of songs for a playlist is usually more important than their exact position within it.

This work was done in collaboration with Andreu Vall from Johannes Kepler University in Linz, Austria. Part of this work was presented as poster at the *ACM Recsys 2017* conference [6].

This chapter is organized as follows. In Section 7.1 we present the related works. In Section 7.2 we present the methods used for automated playlist generation in this work. In Section 7.3 we present the experimental results. Finally, in Section 7.4 we draw the conclusions.

7.1 Background and related works

We briefly review here the most relevant works in the literature of automated playlist generation.

A well-researched approach to automated music playlist generation relies on song content and recommends a next song that is similar to the previous one (e.g., belongs to the same genre or has similar audio characteristics [47, 71, 81, 105]). This is expected to confer coherence to the playlist, but in fact, imposes ad hoc constraints that need not correctly approximate a successful song order.

Playlist generation has also been regarded as a form of Collaborative Filtering (CF), making the analogy that *playlists* are equivalent to *users* [5, 18]. User- and factorization-based CF models require building a playlist profile in order to extend it, which can be a severe limitation in extreme but common tasks such as generating a music playlist from a single seed song.

[1] www.pandora.com

Song-based CF [119] is not limited in this sense. In general, CF models disregard the song order, but it is worth noting that the model presented in [5] accounts for neighboring songs, and the model introduced in [112] (not in the music domain) is aware of sequential behavior.

The Latent Markov Embedding introduced in [24] models playlists as Markov chains. It projects songs into a Euclidean space such that the distance between two projected songs represents their transition probability. The importance of the direction of song transitions is evaluated by testing a model on actual playlists and on playlists with reversed transitions, yielding comparable performance in both cases. Playlists are also treated as Markov chains in [91]. They are modeled as random walks on song hypergraphs, where the edges are derived from multimodal song features and the weights are learned from hand-curated music playlists. The importance of modeling song transitions is assessed by learning the hypergraph weights again, but treating the playlists as sequences of song singletons instead of song transitions. In this case, the performance degrades when the transitions are ignored.

In [54], songs are represented by latent topics extracted from song-level social tags. Sequential pattern mining is performed at the topic level, so that given seed songs, a next topic can be predicted. Re-ranking the results of a CF model with the predicted latent topics is found to outperform the plain CF model.

7.2 Playlist modeling

In this section we describe the models we use for automated playlist generation. We adopt the following approach for every model. Two disjoint sets of playlists are available, one for training and one for test, such that all the songs in the test playlist also occur in the training playlists. Hyperparameter tuning, if necessary, is performed on a validation split that is withheld from the training set. Given one or several songs from a test playlist, a trained model has to be able to rank all the candidate songs according to how likely they are to be the next song in the playlist. At the end of the section we describe the evaluation methodology.

7.2.1 Song Popularity

This model computes the frequency of each song in the training playlists. At test time, the candidate songs are ranked according to their frequency. Thus, the predictions of this model (equivalent to a unigram model –see e.g., [87]) are independent of the current song.

7.2.2 Song-based k-Nearest Neighbors

This is a song-based neighborhood model (see [119]), and its analogous to the Item-kNN model used for session based recommendation in Chapters 4 and 5.

A song s is represented by the set P_s of playlists it belongs to. For each pair of songs s_i, s_j in the training set, we compute their binary cosine similarity

$$\text{sim}(s_i, s_j) = \frac{|P_{s_i} \cap P_{s_j}|}{\sqrt{|P_{s_i}| + \lambda} \cdot \sqrt{|P_{s_j}| + \lambda}}.$$

The value λ is tuned on the validation set and prevents pairs of rarely observed songs from being very similar to one another. This model considers two songs to be similar if they co-occur in the same training playlists, regardless of whether they occupy adjacent positions in them. At test time, the candidate songs are ranked according to their similarity to the current song (previous songs are ignored). Note that the parameter k should define the number of neighbors (i.e., candidate songs) to be considered. However, to make the model maximally expressive, we always consider all the available songs as candidates.

7.2.3 Recurrent Neural Networks

As we have extensively seen in the previous chapters, Recurrent Neural Networks (RNNs) are a class of neural network models particularly suited to learn from sequential data. They have a hidden state that accounts for the input at each time step, while recurrently incorporating information from previous hidden states.

We adopt the same approach proposed in [61] and described in detail in Section 4.1.2. The input to the RNN is now the one-hot vector representation of each song added to the playlist so far, and the output is the likelihood for each song in the dataset to be the next song in the playlist.

The RNN is optimized using AdaGrad [44] with momentum and $L2$-regularization. We also experiment with dropout [128] in the recurrent layer, but none of the final hyperparameter configurations use it. We tune the number of units, the learning rate, the batch size, the amount of momentum, the $L2$-regularization weight and the dropout probability using the validation split. To tune these parameters, we run 100 random search [14] experiments for each of the loss functions enumerated above. The best hyperparameter configuration is chosen according to the validation *Recall@100* (i.e., the fraction of times in which the actual next song is included within

the top-100 ranked candidates), which we consider a proxy of the model's ability to rank the right songs on top positions. The number of training epochs is chosen on the basis of the validation loss. Notice that top-N recommendation lists of this N and even larger (much larger of what any user can actually explore) are common practice in the evaluation of playlist recommender systems [18].

At test time, the RNN model ranks the candidate songs according to the predicted scores. Since it is recurrent, the model is aware of the current song, as well as of all the previous songs in the playlist.

7.3 Experiments

We evaluate the general performance of the models presented in Section 7.2 on two datasets of hand-curated music playlists (described below). For the RNN model, we also explicitly study the importance of the song order in music playlists. As a reference, all the reported results include the performance of a random model that assigns scores to songs uniformly at random, yielding random ranks.

7.3.1 Datasets

The "AotM-2011" dataset [91] is a playlist collection derived from the Art of the Mix[2] database. Each playlist is represented by song titles and artist names, linked to the corresponding identifiers of the Million Song Dataset[3] (MSD) [15], where available.

The "8tracks" dataset is a private playlists dataset compiled from 8tracks,[4] an on-line platform where users can share playlists and listen to playlists other users prepared. Each playlist is represented by song titles and artist names. Since there are many different spellings for the same song-artist pairs, we mimic the AotM-2011 dataset and use fuzzy string matching to resolve the song titles and artist names against the MSD.[5]

We use the MSD as a common name space to identify songs correctly. In both datasets, the songs that could not be resolved against it are discarded, with one of two possible treatments[6]. The first one consists of keeping the original playlists and simply removing the non-matched songs. The

[2] `www.artofthemix.org`

[3] https://labrosa.ee.columbia.edu/millionsong

[4] https://8tracks.com

[5] We adapt the code released in [69] for a very similar task.

[6] Since the reconciliation between the song in the *30Music* dataset and MSD is not possible, we decided to exclude this dataset from the experimental evaluation in order to provide the sufficient guarantees on the quality of the results.

order in the playlist is preserved, but there are skips within them, which we ignore. The second treatment consists of breaking up the original playlists into segments of consecutive matched songs. This way we obtain shorter playlists without skips. For the sake of space, we base our analysis on the first treatment, but experiments on the second treatment yield equivalent conclusions.

A considerable number of playlists in the AotM-2011 contain songs by one or very few artists. In order to study more diverse playlists (which we assume to correspond to a more careful compilation process), we keep only the playlists with at least 3 unique artists and with a maximum of 2 songs per artist. Although the 8tracks dataset is not affected by this issue,[7] we apply the same filters for the sake of consistency. Furthermore, we keep only the playlists with at least 5 songs. This ensures a minimum playlist length, that is required to study the effect of the song position on model performance. Finally, songs included in less than 10 playlists are removed from both datasets to be able to show enough observations of each song to the models.

We randomly assign 80% of the playlists to the training set and the remaining 20% to the test set. Note that full playlists are assigned to either split. At test time, the model deals with playlists that were never seen before. As in any recommendation task blind to item content, the songs that only occur in test playlists need to be removed because they can not be modeled at training time.

The filtered AotM-2011 dataset includes 17,178 playlists with 7,032 songs by 2,208 artists. The filtered 8tracks dataset has 76,759 playlists with 15,649 songs by 4,290 artists. Tables 7.1 and 7.2 report the distribution of unique songs per playlist, unique artists per playlist and song frequency in the datasets.

7.3.2 Model parameters

The description of model parameters for each dataset used in our experiments follows.

AotM-2011 dataset The song-based k-nearest neighbors model (k-NN) uses $k = 6{,}258$ neighbors (all the songs in the test playlists) and a smoothing factor $\lambda = 100$. The RNN has 200 units, a learning rate of 0.01, mini-batches of 16 playlists, a momentum coefficient of 0.5 and an $L2$-regularization weight of 0.1. The RNN that is trained on shuffled playlists has 200 units, a

[7] The terms of use of the 8tracks platform require that no more than 2 songs from the same artist or album may be included in a playlist.

AotM-2011		min	1q	med	3q	max
Training set	Songs per playlist	5	6	7	8	33
	Artists per playlist	3	5	7	8	32
	Playlists per song	1	6	10	16	201
Test set	Songs per playlist	5	6	7	8	34
	Artists per playlist	3	5	7	8	34
	Playlists per song	1	2	3	5	50

Table 7.1: *Descriptive statistics for the playlists in the AotM-2011 dataset. "Playlists per song" indicates the number of playlists in which each song occurs.*

8tracks		min	1q	med	3q	max
Training set	Songs per playlist	5	5	6	7	46
	Artists per playlist	3	5	6	7	41
	Playlists per song	1	7	12	24	1,860
Test set	Songs per playlist	5	5	6	7	45
	Artists per playlist	3	5	6	7	40
	Playlists per song	1	2	4	7	458

Table 7.2: *Descriptive statistics for the playlists in the 8tracks dataset. "Playlists per song" indicates the number of playlists in which each song occurs.*

learning rate of 0.025, mini-batches of 32 playlists, a momentum coefficient of 0.4 and an $L2$-regularization weight of 0.05.

8tracks dataset The song-based k-NN model uses $k = 14{,}299$ neighbors (all the songs in the test playlists) and a smoothing factor $\lambda = 100$. The RNN has 200 units, a learning rate of 0.025, mini-batches of 64 playlists, a momentum coefficient of 0.3 and an $L2$-regularization weight of 0.02. The RNN that is trained on shuffled playlists has 200 units, a learning rate of 0.025, mini-batches of 64 playlists, a momentum coefficient of 0.5 and an $L2$-regularization weight of 0.01.

7.3.3 Evaluation procedure

Each trained model is evaluated by repeating using the sequential evaluation procedure described in Section 2.6.1. We show the model the first song in a playlist. It then ranks all the candidate songs according to their likelihood to be the second song in that playlist. We keep track of the rank assigned to the actual second song and of the fact that this was a prediction for a song in second position. We then show the model the first and the

second actual songs. The model has to rank all the candidate songs for the third position, having now more context. In this way, we progress until the end of the playlist, always keeping track of the rank assigned to the actual next song and the position in the playlist for which the prediction is made.

A perfect model would always rank the actual next song in the first position. A random model would, on average, rank the actual next song approximately in the middle of the list of song candidates. An extremely poor model would rank the actual next song in the last position. Note that the actual rank values depend on the number of candidate songs available.

Previous research has often summarized the ranking results in terms of Recall@N, where N is the length of the list of top next recommendations (see e.g., [18, 54]). However, the proposed evaluation setting may be too pessimistic in the music domain [90, 104], where songs other than the actual one may serve as valid playlist continuations. As a consequence, long lists of next-song candidates are needed to observe the model behavior.

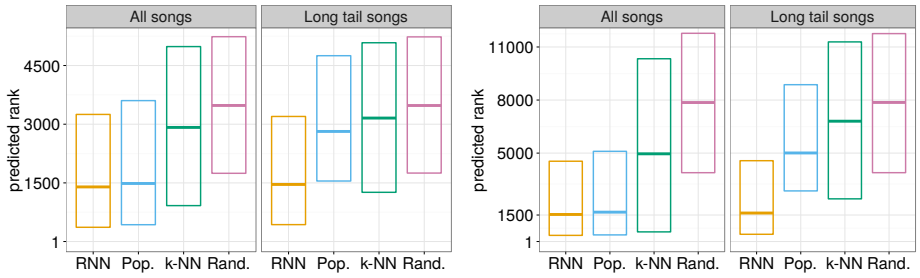
In order to better observe the performance of each model, we opt for analyzing the full distribution of predicted ranks (the lower the better), summarized by the first quartile, the median and the third quartile rank values (see Figures 7.1 and 7.2). This approach also facilitates the comparison of different models.

7.3.4 Model performance and the long tail

We first turn our attention to the general performance of the playlist models, depicted in Figure 7.1. When we consider all the songs (left-hand side of the figures), the RNN and the popularity-based model perform comparably well and significantly better than the song-based k-Nearest Neighbors (k-NN) model. This is at first surprising (given the simplicity of the popularity-based approach), but it can be explained by the impact of a small number of very popular songs in both datasets (see Tables 7.1 and 7.2). Interestingly, if the 10% most popular songs are ignored, the inability of the popularity-based and the song-based k-NN models to correctly rank next songs becomes clear, with performances close to random.

This is a clear reminder that we cannot ignore the long-tailed nature of datasets in the music recommendation domain [23], with consequences regarding the importance of song order in music playlists:

> ***Observation:*** *Under our experimental setting, if the long tail of non popular songs is not considered separately, a simple popularity-based model, that completely neglects the order, performs comparably to an RNN model.*



(a) *AotM-2011 dataset* **(b)** *8tracks dataset*

Figure 7.1: *Distribution of ranks predicted by each model. The boxplots indicate the first quartile, median and third quartile ranks (lower is better). Left: All songs are considered. Right: The 10% most popular songs are excluded (only the long tail of non popular songs is considered). "Pop." and "Rand." correspond to the popularity-based and the random models, respectively. The scale of the $\mathbf{y}$-axis relates to the number of song candidates in each dataset.*

7.3.5 Song Order Randomization

In the following experiments the RNN has to model actual playlists where the song order has been randomized.

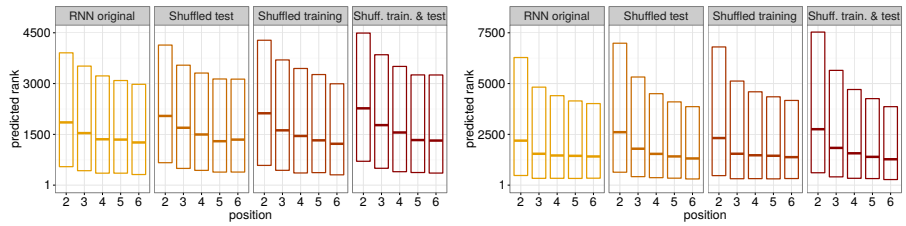
1. *Shuffled test*: the RNN is trained on actual training playlists and evaluated on shuffled test playlists. This is a *weak* check because if song order is relevant, the RNN can potentially exploit it at training time.

2. *Shuffled training*: the RNN is trained on shuffled training playlists[8] and evaluated on actual test playlists. This is a *strong* check because if song order is relevant, we now make sure to break it. This is to say that the RNN can not learn from the order at training time.

3. *Shuffled training and test*: the RNN is trained on shuffled training playlists and evaluated on shuffled test playlists. We conduct this experiment for completeness.

The results are represented in Figure 7.2. We separate the rank distributions depending on the position in the playlist for which the prediction is made. Thus, we obtain a more fine-grained view on the distributions and can observe the impact of knowing the preceding songs on next-song predictions. As a reference, we include the performance of the RNN model on playlists in original order.

[8]The RNN hyperparameters are re-tuned for this and the next experiment, to ensure that the performance of the model is not compromised just as a consequence of modifying the training data.



(a) *AotM-2011 dataset* **(b)** *8tracks dataset*

Figure 7.2: *Distribution of ranks predicted by the RNN for original and shuffled playlists, grouped by the position for which the prediction is made. The boxplots indicate the first quartile, median and third quartile ranks (a lower rank is better). The scale of the y-axis relates to the number of song candidates in each dataset.*

We observe that the RNN performs comparably well for playlists in original order and for any other shuffling experiment, including those in which we train the RNN on shuffled playlists where the order is for sure broken. This indicates that the song order does not inform the RNN to make better next-song predictions.

We also observe that the RNN performance improves (i.e., it assigns lower ranks to the actual next songs) as more knowledge of the preceding songs is available. This is a reasonable behavior for an RNN, which informs its hidden state as it progresses through the playlist. It suggests that the choice of songs is highly informative for next-song predictions.

Observation: *Under our experimental setting, when using an RNN model, knowledge of the preceding songs is informative for next-song predictions, but their exact order is not.*

7.4 Conclusions

In this chapter, we have explicitly investigated the importance of song order in hand-curated music playlists. We have conducted offline experiments with different playlist models (including state-of-the-art sequential models) on two independent datasets. Our findings indicate that the song order is not informative for next-song predictions. This is evident in heavily popularity-biased datasets, where predicting the next-song based on popularity alone performs comparably to a sophisticated sequence-aware RNN model. Once the trivially predictable popularity tracks are removed, the advantage of the more complex RNN model becomes evident. We have challenged RNN models, by submitting them to different configurations of shuffled playlists.

Our results show that, in this specific problem, sequence-aware models cannot exploit the order of songs seen during training to make better next-song predictions in the evaluation. However, we have found that the knowledge of preceding songs is indeed informative to identify which song comes next. This further underpins the robustness to unprecedentedly seen sequences of the RNN-based models, which is the family of algorithms subject of this dissertation.

Since playlist generation does not seem suitable for further investigation on sequence-aware methods, in the next chapter we will explore another application scenario in music recommendation, that of sequential listening prediction. We will present a new large scale dataset that is openly accessible to researcher and practitioners interested in the topic.

CHAPTER 8

A dataset for large-scale music listening recommendation

Music recommender systems propose interesting music to a specific user. Differently from many other recommendation domains - such as movie recommendation - items in the music domain are short in duration and are usually consumed many times by the same user.

Even though users have general preferences about the music they like, for example in terms of genres or artists, music listening is a strongly contextual experience. Playlists are somehow "curated" sequences of songs that can be played several times by the user and fulfill some quality constraints in terms of diversity and coherence between songs, for example at genre, artist or acoustic level. For such reason, in the previous chapter we had to remove from our datasets the playlists that did not respect certain minimal quality constraints to ensure the quality of the obtained results.

Music listening, instead, is about the "raw" signals (play events) generated by users while they exploring the catalogs of available songs. Music listening behavior is more prone to be influenced by other contextual factors, such as mood, that can be hardly elicited from users. To overcome such limitation, we propose to use the current listening session as a proxy

to infer the user intent and to predict the tracks will likely listened in the remainder of her listening session. However, no publicly available dataset allows to address this research need completely.

To address this research need, in this chapter we present the *30Music* dataset, a new large-scale dataset of music listening sessions collected from Last.fm. To the best our knowledge, there is no other publicly available dataset of music listening sessions to date. By releasing this new dataset, our intent is to foster the research into session-based music recommendation and contextual playlisting. This is an emerging research field, as has been highlighted also by other surveys in the field [18].

The dataset was presented as a poster paper at the conference *ACM Recsys 2015* and in the *Large Scale Recommender Systems (LSRS)* workshop of the same conference.

The outline of the chapter is the following. In Section 8.1 we briefly describe the other publicly available datasets. In Section 8.2 we describe the dataset generation process and analyze its salient characteristics. Finally, in Section 8.3 the conclusions are drawn.

8.1 Background and Related Work

There exist a number of publicly-available music datasets, used in several music experiments. The most popular and relevant ones are shown in Table 8.1.

Most datasets provide content information (e.g., metadata, tags, acoustic features), but only a few report some user-system interactions (e.g., ratings, play events) useful to profile users and to experiment with personalization tasks.

The Million song dataset (MSD) [16] is a public collection well-known for its size. In fact, it contains audio features (e.g., pitches, timbre, loudness, as provided by the Echo Nest Analyze API[1]) and textual metadata (e.g., Musicbrainz[2] tags, Echo Nest tags, Last.fm tags) about 1M songs (related to 44K artists).

Celma [23] has published two music datasets collected from Last.fm API: 1K-user and 360K-user. The smallest one - 1K-user dataset - contains the user listening habits (20M play events) of less than 1K users. On the other hand, the biggest one - the 360K-user dataset - collects the information about 360K users, but it does not have any listening data other than the number of times a user has listened to an artist. Data are provided as

[1] http://the.echonest.com/
[2] https://musicbrainz.org/

downloaded from the Last.fm API.

Yahoo! Labs have released several music datasets[3], with different sizes and content. For instance, the R1 dataset provides 10M ratings of musical artists; the R2 dataset contains 717M ratings of 136K songs given by 1.8M users, where each song is accompanied by artist, album, and genre attributes; the R9 dataset publishes track metadata collected by monitoring the track plays of more than 4K Internet radio stations during a period of 15 days; finally, the C15 dataset is a very large dataset providing four types of user ratings (tracks, albums, artists, and genres) and being used in recommender system challenges [43]. Unfortunately, none of the datasets contains user play events.

Some datasets have been extracted from microblogs. For instance, the #nowplaying dataset [147] comprises about 50M listening events of 4M users about 1.3M tracks, extracted from Twitter messages containing the hashtag '#nowplaying'. Similarly, the Million Musical Tweets Dataset [55] and the MusicMicro dataset [121] contain a set of geolocalized music listening events collected by monitoring a portion of Twitter messages, filtering special music-related hashtags, and maintaining only tweets containing a location. In addition, MusicBrainz was used to verify the existence of the track referenced by the tweet. Finally, the data was enhanced with further information using the Yahoo! Place Finder API (linking to locations), the Last.fm API (adding tags), Allmusic.com (adding user mood information), and the 7digital API (retrieving 30-second audio track).

The Art of the Mix Playlist dataset[4], collected from the artofthemix.org website, consists of 29K user-contributed playlists, containing 218K distinct songs for 60K distinct artists. However, there are not user listening events.

Finally, the USPOP'02 dataset [13] is typically used to experiment with tasks exploiting audio features. In fact, it mainly contains the acoustic features about 8K tracks from 400 popular artists, together with a set of tags for each artist.

In summary, the available datasets have the following four main limitations:

1. They have only either implicit play events (*Celma 1K*, *Celma 360K*, *#nowplaying*, *MMTD*, *Music Micro*) or explicit preferred tracks or artists (*Yahoo! R1*, *Yahoo! R2*, *Yahoo! C15*, *artofthemix*). There are no datasets with both the type of information.

[3] http://webscope.sandbox.yahoo.com/catalog.php
[4] http://labrosa.ee.columbia.edu/projects/musicsim/aotm.html

2. Datasets with play events do not provide the duration of the events. Therefore, with these datasets it is not possible to make a distinction between skipped songs (i.e., negative implicit feedback) and full listened songs (i.e., positive implicit feedback).

3. There are no datasets in which play events are organized into sessions of contiguous and ordered songs. As the sequence of play events is meaningful for the user, without this information it is not possible to evaluate the ability of a music recommender system to create playlists matching the user taste and context.

4. There are no datasets (with the exception of *artofthemix*) which provide user generated playlists. The number of playlists in *artofthemix* is small (29K) and the dataset does not provide any other information about user interactions or explicit user preferences.

8.2 The *30Music* dataset

In this chapter we introduce the *30Music* dataset, a collection of listening and playlists data retrieved from Internet radio stations and from over 600 music players (including Rdio, Spotify, Clementine, Amarok, MusicBee) through the Last.fm API.

Data refers to four different types of user interactions – listened tracks, listening sessions, user-created playlists, explicit users' likes on tracks – enriched with user and item metadata, such as user demographics and track attributes (artists, albums, social tags).

In this chapter we describe the dataset creation process, its content, and its possible uses. Attractive features of the *30Music* dataset that differentiate it from existing public datasets include, among the others, (i) the user listening sessions complete of contextual time information, and (ii) the user playlists, key information to experiment with the task of modeling user taste and recommending sequences of tracks. Implicit user's interests can be derived by joining the duration of each listening session with the length of the corresponding music track, in order to detect "skipping" behaviors.

8.2.1 Dataset creation

The *30Music* dataset has been obtained via Last.fm public API[7]. Last.fm is a music website that offers several social networking features (e.g., a

[5] Ratings given to artists
[6] Playcount of artists
[7] `http://www.last.fm/api`

collaborative wiki system, social tagging, etc.) and can recommend artists to the users. Last.fm provides, among the others, free API to track details of user listening sessions.

In the case a user has connected his player to his Last.fm account (which typically simply implies setting the Last.fm username and password), the player transfers information on the user listening activity via the Last.fm's *scrobbling* API. Scrobbled messages contain information such as track metadata (e.g., title, artist, and album) and the event timestamp. Last.fm stores the data in the same format as it is transmitted from the player; consequently, scrobbling events can report wrong or incomplete metadata (e.g., a misspelled title). Scrobbled information are publicly available through the Last.fm API. Unfortunately, there is no way to identify which player generated a specific scrobbled event.

Scrobbling allows Last.fm to collect the listening activity of the users on any online music player in a way transparent for the user; it does not require any change in the way users listen to their music. The scrobbling events are registered by Last.fm to collect statistics on the usage of their services and generate personalized recommendations to users [9].

It is worth noting that only listening events are recorded, while pause/skip events are not scrobbled from the user player to Last.fm, as well as any playlist or explicit preference defined or expressed in the player. The main way for a user to create a playlist in Last.fm is to access to the website; similarly, the user can express explicit preferences ('love') about tracks directly in the website.

To build the *30Music* dataset, we started from a list of 2M Last.fm usernames from the Chris Meller dataset [8]. This dataset was obtained using iteratively the `User.getFriends` call starting from 'Chris Meller' as seed username and then crawling his friends in the Last.fm social network. Given the list of users, we used the `User.getPlaylists` call to collect the list of playlists created by each user. For each retrieved user playlist, we call the `Playlist.fetch` to obtain the single tracks composing the playlist. Users with at least one playlist were about 45K. Starting from this set of users, we called the `User.getRecentTracks` to retrieve the user listening events over a 1-year time window (events are crawled from Mon, 20 Jan 2014 09:24:19 to Tue, 20 Jan 2015 09:24:19).

Raw playlists and user listening events have been enriched with additional information about users and tracks. For each user, we the following user demographics with `User.getInfo`: country, age, gender, number

[8] `https://opendata.socrata.com/Business/Two-Million-LastFM-User-Profiles/5vvd-truf`

of playlists, total playcount, registration date and whether she is a premium user of Last.fm. Similarly, we retrieved the following track metadata with `Track.getInfo`: duration, global playcount, artist/album/-track MusicBrainz [9] identifiers, album title and Last.fm social tags.

Data processing

We have performed some basic data cleaning on the raw data. For example, users with wrong metadata or that are not recognized by the Last.fm API were removed (this reduced the original set of 2M users to one half). We also performed applied some basic track deduplication strategies, such as tracks with exact artist and track names were associated to an unique identifier. However, we did not unify mispelled tracks or tracks with multiple versions (e.g., studio, live, unplugged versions *etc.*).

Finally, we defined a new entity, the *user play session*, as an ordered list of play events that are assumed to be consequently listened by the user with no interruptions. We define a play event to be part of a session if it occurs no later than 800 seconds after the previous user play event. The choice of the threshold was made because the empirical distribution of the play time exhibits a drop close to 800 seconds. Anyways, it is still possible to define other partitioning into sessions from the data using other thresholds.

30Music dataset format

The *30Music* dataset is released in accordance with the Idomaar data format [117], a multigraph representation oriented to recommender system evaluation that explicitly represents *entities* (i.e., nodes) and *relations* (i.e., edges). An entity models any object that can be recommended (e.g., a user can be recommended a track, an existing playlist, an artist, or he can be even suggested a tag). A relation models a link between two (or more) entities (e.g., the play event of a user about a track at a certain time). We defined the 6 different entities and 3 types of relations.

Entities:

- 45K *users*. A Last.fm user is identified by his username. Users are stored together with additional information about gender, age, country, number of scrobbled events, number of created playlists, the time they registered to Last.fm service, and whether they are premium users.

[9] `https://musicbrainz.org/`

- 5.6M *Tracks*. A Last.fm track is identified by the title and the artist. Tracks report additional information such as duration, social tags, album and their respective track MusicBrainzID (available for 1,469,606 tracks).
- 57K *Playlists*. A Last.fm playlist is an ordered lists of tracks defined by a specific user. A playlist has a Last.fm identifier, a title and it contains the information about the user that has created it with the related timestamp, and the tracks it consists of.
- 595K *Artists*, represented by their Last.fm name and their respective artist MusicBrainzID (available for 109,285 artists).
- 217K *Albums*, represented by their Last.fm name and the album MusicBrainzID for all of them.
- 277K *Tags*, the social annotations explicitly provided by Last.fm users to specific tracks, their value and the Last.fm url. Tags are available for 1.5M tracks with 3.7 tags/track on average.

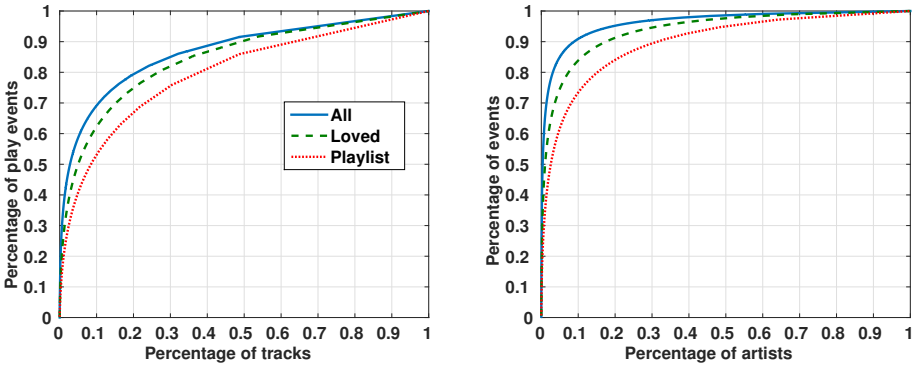
Relations:

- 31M *User play events*. Each relation represents the play of a track by the user in a specific timestamp.
- 2.7M *User play sessions*. A session, in accordance with the definition of the previous subsection, contains the time the session starts, the number of tracks, the ordered list of each played track together with the start time (related to the begin of the session), the play time, the play ratio (i.e., the playtime divided by the track duration).
- 1.7M *User love preferences*. A user love preference reports the explicit preference given by a user about a specific track.

8.2.2 Dataset analysis

This section presents a statistical analysis of the *30Music* dataset. The dataset contains 31,351,954 play events organized into 2,764,474 sessions (an average of 11 play events per session). The dataset contains also 1,692,924 explicit ratings (loved tracks), with an average of 33 ratings per user, and 279,351 user-created playlists. Due to the session generation process, the track duration is not available for the last track of each sessions.

Figure 8.1a plots the empirical cumulative event distributions as a function of the number of tracks (in percentage). Tracks in the horizontal axis



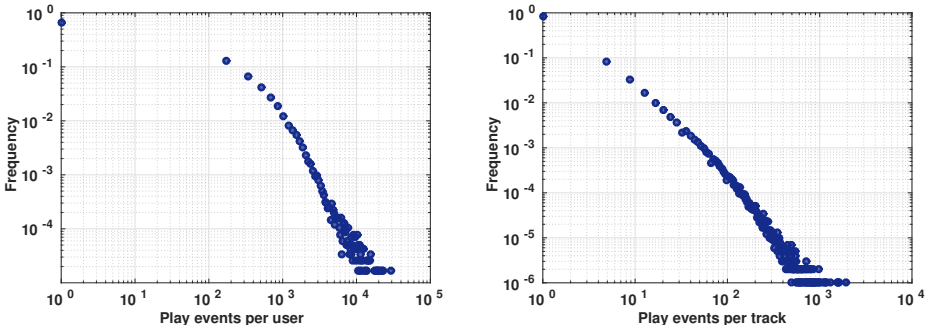
(a) *Tracks* **(b)** *Users*

Figure 8.1: *Cumulative distribution of play events for tracks and artists*

were ordered according to their popularity, most popular to the left. The figure plots the same distribution for two subsets of the play events: play events of "loved" tracks (i.e., tracks for which the user expressed an explicit preference) and play events of playlist tracks (i.e., tracks played from a playlist). We observe that play events present a moderately long-tail distribution, since $\sim$20% of the most popular tracks collect $\sim$80% of the play events. This long tail effect is mitigated when analyzing preferred tracks (i.e., loved tracks and tracks in the playlists). We observe that the same percentage of play events (80%) involves twice the tracks (40%) when considering tracks in the playlists. We can deduce that user's explicit preferences span a wide set of tracks, but listening habits are strongly biased toward the short-head of the most popular tracks.

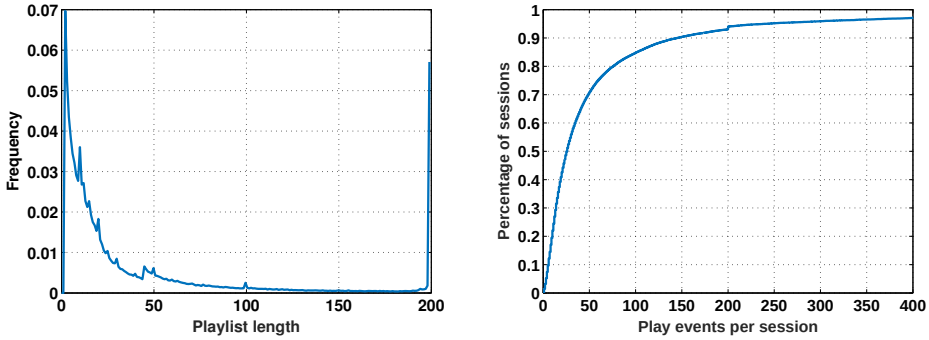
In the analysis over artists (Figure 8.1b) we can notice stronger long-tail effect than what observed for individual tracks, e.g. the 20% most popular artists collect more than 95% of the play events. Similar considerations to the previous case on loved and playlist tracks also apply for artists. We can deduce that users have preferences spanning many different artists, but their listening behaviour is strongly biased toward the short-head of the most popular artist.

Figure 8.2 depicts the distribution of the number of ratings per user (Figure 8.2a) and per track (Figure 8.2b). Both distributions are clearly power-law behavior. There is huge number of spurious users with a single play event, then all users have more than $\sim$100 play events each, with a tiny fraction ($\sim$1%) of very active users with $\geq$ 1000 play events each. The distribution of the play event per track was already analyzed in the previous



(a) *Users* **(b)** *Tracks*

Figure 8.2: *Empirical distribution of the number of play events per user and track*



(a) *Playlist lengths* **(b)** *Number of events per session*

Figure 8.3: *Probability distribution of the playlist lengths and cumulative distribution of the number of events per session*

plots, and here the power-law distribution of the play events per track is highlighted even further.

Figure 8.3a shows the number of playlists in the dataset as a function of their length. As one would expect, the number of available playlists decreases with their length. ThWe can also observe that many playlists have length of 200 in correspondence to the limit posed by Last.fm itself.

Figure 8.3b refers to play events and shows that 70% of the sessions have at most 50 tracks. Note that the small jump at length 200 events, likely correlated with playlists of the maximum available length that are listened completely by users.

Finally, in Figure 8.4 we show the distribution of the playtime and duration of the tracks in the dataset. Both distributions are centered at 200-250 seconds ($\sim 3 - 4minutes$) which are reasonable track durations and play-

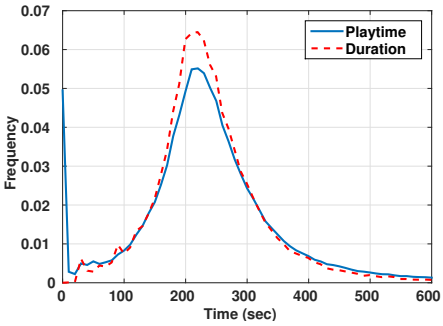


Figure 8.4: *Distribution of track playtimes and duration.*

times. Frequency drops significantly after 500-600 seconds, slight before the 800 second threshold used to generate sessions. Notice that the peak below 5 seconds is likely due to spurious scrobbling events and cannot be straightforwardly imputed to skip events.

8.3 Conclusions

In this chapter we have presented *30Music* dataset, a novel large-scale dataset of music listening records crawled from Last.fm. We delved in depth into the characteristics of the dataset and compared it against the other publicly available datasets in the field. From a search on Google Scholar[10], this dataset has been already used in ~10 publications or thesis works to date. We believe this dataset has already helped, and will keep helping, the research on the modeling of music listening habits.

[10]https://scholar.google.com

		MSD	Celma 1K		Celma 360K		Yahoo! R1	Yahoo! R2	Yahoo! C15	artoftthemix	#nowplaying	MMTD	Music Micro	*30Music* dataset
	Sources	(a)		(b)				(c)		(d)	(e)		(f)	(b)
Features	Release	2011	2010		2010		n/a	n/a	n/a	2003	live	2013	2012	2015
	Span	n/a	4y		n/a		1m	5y	10y	1m	4.5y	18m	1y	1y
	Users	-	1K		360K		2M	1.8M	-	1M	5M	15K	136K	45K
	Tracks	1M	1M		-		-	136K	625K	218K	764K	134K	71K	5.6M
	Artists	44K	180K		160K		9.4K	100K	-	60K	95K	25K	19K	595K
	Playlists	-	-		-		-	-	-	29K	-	-	-	280K
	Track info	x	-		-		-	x	-	-	x	-	-	x
	Acoustic	x	-		-		-	-	-	-	-	-	-	-
	User info	x	x		x		-	-	-	-	-	-	-	x
Interactions	Play events	-	20M		-		-	-	-	-	63M	1M	600K	31M
	Playtime	-	-		-		-	-	-	-	-	-	-	x
	Sessions	-	-		-		-	-	-	-	-	-	-	2.7M
	Ratings	-	-		-		12M[5]	717M	262M	-	-	-	-	1.7M
	Play count	-	-		17M[6]		-	-	-	-	-	-	-	-

(a) EchoNest, MusicBrainz, Last.fm
(b) Last.fm
(c) Yahoo!
(d) Art of The Mix
(e) Twitter
(f) Twitter, MusicBrainz, Yahoo! Place Finder, Last.fm, 7digital, allmusic.com

Table 8.1: *Comparison of different music datasets.*

Part V

Conclusions

CHAPTER 9

Conclusion and perspectives

In this thesis, we have developed a series of novel algorithms for sequence-aware recommendation. First, we have conducted an exhaustive survey of the state of the art in order to characterize the problem and identify the overlooked research aspects in the existing works. We then focused on a specific task, that of session-based recommendation, that can be effectively tackled by sequence-aware recommenders to enhance the quality of the recommendations above the existing methods. We have validated the importance of personalized recommendations in a specific session-based through a user study. We developed a novel recommender system based on parallel-RNNs, capable of exploiting both item identifiers and rich item features to improve the quality of the recommendations in session-based scenarios with new users. We developed a novel session-aware recommender based on Hierarchical RNNs that jointly models the user activity between and across sessions, providing an unified solution to personalize recommendations for returning users. We then investigate sequence-aware recommendation in music. We employed sequence-aware recommenders to model the evolution of the musical tastes of the user over time. After that, we investigated the importance of the order of the songs on the quality of automated playlist generation. Finally, we presented a novel large-scale

dataset for session-based music recommendation, with the goal of helping the research in this field.

In this chapter we first summarize and conclude the work conducted in this thesis. We then present the future research directions for expanding this work.

9.1 Summary and contributions

A discussion on the main contributions of this work with respect the research goals established in Chapter 1 follows.

9.1.1 Characterizing Sequence-Aware recommender systems

In Chapter 2 we presented a novel and exhaustive survey of the state of the art in sequence-aware recommender systems. We have first provided an intuitive definition of the problem. Then, we have characterized sequence-aware recommenders in the terms of their inputs, outputs and computational tasks. We have analyzed their relations with respect to other related areas in Recommender Systems, by highlighting similarities and differences. We then described the main tasks and subtask for which sequence-aware recommenders are usually employed. We noticed that context-adaptation is the main task that can be addressed with sequence-aware recommender systems, driven also by the emergent applications in session-based and session-aware recommendation. Finally, we provided a taxonomy for the existing algorithms evaluation procedures for sequence-aware recommenders.

From this analysis, we were able to identify several overlooked research aspects that needed to be addressed. We focused on session-based and session-aware recommendation, because of their stronger connections with many real-world applications, and because they are relatively less explored aspects in the literature of sequence-aware recommenders. In particular, we were able to identify the complete absence of feature-based sequence-aware models and of "holistic" session-aware recommenders, as well as the lack of user studies in the field.

9.1.2 Session-based Hotel Recommendation: a User Study

We first run a user study to assess the influence of personalized recommendations in session-based recommendation. In Chapter 3 we build a simple yet effective sequence-aware recommender based on implicit elicitation and time decay, and proved that personalization is indeed effective even in contexts in which no past information on the user interests is available.

Specifically, we run our user study in a strong popularity biased domain, that of hotel booking, and evaluated the perceived impact of recommendations in a scenario of limited availability of hotels ("when the best are gone"). Our findings show that personalized recommendations are capable of keeping constant user satisfaction even when best available options for the user are no longer available. From this experiment, we decided to investigate over more sophisticated sequence-aware recommenders, that can go beyond what is achievable by combining traditional collaborative filtering and content-based algorithms with the implicit elicitation mechanism.

9.1.3 Feature-rich session-based recommendation with Recurrent Neural Networks

From the analysis of the state of the art, we advise the lack of sequence-aware recommenders capable of exploiting rich item features. The vast majority of the research focuses on the extraction of patterns from sequences of item identifiers, but valuable information can be undoubtedly extracted from the content of the products visualized by the user. This can in turn provide essential information in new-user scenarios.

In Chapter 4, we have presented a novel sequence-aware recommender based on parallel-Recurrent Neural Networks (p-RNNs). We proposed p-RNN architectures that can leverage the added value of multiple item representations, such as item identifiers and dense (or sparse) item feature descriptors. We devised alternative training strategies (alternating, residual and interleaving training) to fit these architectures. The proposed architectures and training methods outperformed the baselines by large margins. Nevertheless, the improvements were mostly in Mean Reciprocal Rank, meaning that item features could be effectively exploited by the model to order the items in the recommendation list better, but not to retrieve more interesting items for the user, since Recall is only marginally better.

9.1.4 Personalizing Session-based Recommendations with Hierarchical Recurrent Neural Networks

We also advise the lack of unified solutions for personalized session-based (or session-aware) recommendation for returning users. The relatively few existing solutions decouple the long-term modeling of the user interest from the short-term (session-based) modeling of the user activity. This in turn requires periodic updates of the long-term models to keep track of the evolution of the user interests over item.

To address this and other issues, in Chapter 5 we have presented a unified framework that jointly models the user activity within and across sessions. Our framework uses HRNNs, in which the lower-level of the hierarchy models the activity of the user within each session, and the upper-level models the evolution of the user's interests across sessions. The upper-level is also used to initialize the representation of the lower-level, hence to provide the necessary personalization capabilities to the session-based recommender. Our experimental results show the superiority of our approach with respect to "pure" session-based recommendation and to trivial personalization strategies such as session-concatenation in all the evaluated metrics. We could also show that the quality of the recommendation improves with the length of the history of the user both within and across sessions, further justifying the need for personalized models for returning users also in strongly session-bounded domains, such as video streaming recommendation. These results underpin the validity of our approach to session-aware recommendation.

9.1.5 Modeling Musical Taste Evolution with Recurrent Neural Networks

We then investigate sequence-aware recommendation in music recommendation. In Chapter 6 we use Recurrent Neural Networks to model the evolution of user tastes over time. We consider the task of music station recommendation as a proxy for the user musical interests, and compare RNN-based models against other sequence-aware methods. The experimental results show that RNN leads when enough historical data is available, since it can effectively predict the next stations added by the user with greater accuracy. Moreover, RNN-based methods look less prone to popularity biases than all alternatives, allowing for a much broader exploration of the catalog of genres, artists and tracks available. We design a learning-to-rank experiment to leverage this property. The experimental result show significant improvements in the ranking quality when the top-10 recommendations generated by RNN are added to the pool of candidate recommendations to re-rank.

9.1.6 Automated Playlist Generation with Recurrent Neural Networks

In Chapter 7 we addressed one long-debated issue in music recommendation, that of the importance of the order of songs in automated playlist generation. In our survey of the state of the art, we could notice that order constraints of different degree were considered in the literature of music

recommendation. However, there is little consensus on whether the order of songs matters to the final quality of a playlist.

We therefore exploited the knowledge built on sequence-aware recommenders based on Recurrent Neural Networks to shed some light on this issue. We designed an experiment aimed at verifying whether the order of the songs seen by the model in training was of any importance in predicting the songs in a held-out set of playlists. The experimental results confirmed that the order of songs is not relevant. Moreover, our experiments showed that RNN are capable of building meaningful sequence representations even when the order of the songs in the playlist is shuffled, showing unexpected robustness to unseen sequences of songs.

9.1.7 A dataset for large-scale music listening recommendation

In Chapter 8 we presented a new large-scale dataset for music recommendation, the *30Music* dataset dataset. As our analysis of the existing datasets showed, our dataset has unprecedented size and it is one of the few to provide listening sessions out-of-the-box. We described the salient characteristics of *30Music* dataset, and discussed about the research opportunities that it opens.

9.2 Future works

This thesis, as we have analyzed previously, presents several novel aspects of sequence-aware recommendation. There are still open directions that we could not cover in this work and that we would like to highlight here.

9.2.1 User studies

First, *user studies* are required to understand the actual effectiveness of some of the proposed strategies. A first attempt was done in Chapter 3. However, a larger pool of users is required to evaluate the online validity of the algorithmic approaches presented in this thesis, as well as other application domains.

In particular, in our experiments on p-RNNs analyzed in Chapter 4, we noticed that the RNN configurations based only on item features return meaningful results, even if performed poorly in the offline evaluation. For example, in many cases the visual appearance of the recommended items is close to the actually clicked item but not exactly the same. The offline evaluation can therefore be overly pessimistic for these models, and online user studies are required to investigate more on this issue.

9.2.2 Item cold-start

A related open issue is that of *item cold-start* with sequence-aware recommenders. Similar to what occurs in traditional collaborative filtering, the vast majority of the existing methods for sequence-aware recommendation require items to occur a certain minimum number of times in the training set to be considered by the recommender. For example, very "long-tail" items that occur rarely in the training sequences will be blocked by strict minimum support thresholds in frequent pattern mining based models, or will be associated with noisy transition probabilities and output probabilities in transition models like RNNs. This issue is further amplified for new, cold-start items that are unseen during training. Continuous item cold-start is an important issue in domains with high item turn-over rates, in which new items are constantly added and old items quickly becomes obsolete, like for news and classified advertisement, for example.

Parallel-RNNs can be readily used to include new and long-tail items in the pool of recommended items, given their ability of learning and exploiting sequential relations between item features. In our experiments, we have shown the effectiveness of p-RNNs when item features that are directly extracted from raw content are used. This can in turn lead to significant advantages over methods based on "high-level" features such as metadata, since such high-quality descriptors may not be readily available for every new content added to the system.

9.2.3 Multiple interaction types

Another open research direction is the inclusion of *interaction types* in our sequence-aware models. In our contributions we have considered a single interaction type (e.g., the user clicks on an advertisement, views a video or listens to a song). However, in real scenarios users can have multiple types of interactions with the user interface (e.g., playing, replaying or skipping a song). We presented a preliminary analysis on how multiple signals can be used with traditional recommenders in session-based scenarios in Chapter 3.

However, little research has been done to date on how to exploit multiple feedback types in a sequence-aware fashion. Such feedback can be extremely useful in detecting the intent of the user in session-based scenarios, for example. We also believe that this problem has received little attention from the research community also due to the lack of publicly available datasets that provide such type of information (to our knowledge, the XING dataset is the first dataset to do this). Interaction types, item and user fea-

tures could be jointly considered to extend our HRNN model into a holistic user model that dynamically evolves the user representation over time and promptly adapts to the user's interests within each session, also on the basis of the characteristics of the items the user usually interacts with.

9.2.4 Address subtasks

From the analysis of the state of the art we conducted in Chapter 2, we noticed that many subtasks distinct from context-adaptation can be targeted by sequence-aware recommenders. For example, the proposed HRNN model "implicitly" incorporates the individual trends in the user interests by modeling its long-term activity through time. From our experiments, the same model showed a certain degree of adaptability also to repeated user behaviors. However, further investigation is needed to verify the validity of these assumptions. New sequence-aware models can be also devised to target these specific subtasks, which can be highly relevant according to the application scenario [75].

9.2.5 Session-based music recommendation

Contextual music recomemndation is an emerging research field [18]. In this work we have presented a dataset aimed at helping this research, but more datasets and the further investigation is certainly needed to develop new sequence-aware algorithms suitable for this problem.

9.2.6 Connections with other areas in Recommender System research

We envision some connections with other research areas in RS research. A completely unexplored area is that of cross-domain sequence-aware recommendation. Prior studies has shown that user and item cold-start issues in one domain (e.g., video) can be alleviated by transferring knowledge from auxiliary domains (e.g., music, book) in which the user-item dynamics are better known [134]. This is possible especially when it is possible to establish a connection between the source and auxiliary domains [33]. However, cross-domain recommendation has been studied only in the terms of the traditional user-item matrix framework. We hypothesize that useful knowledge could be transferred across domains also from user interaction logs, and novel cross-domain techniques can alleviate the item and user cold-start issues in sequence-aware recommendation, that can be solely solved with content-based methods up to now. For example, the sequential listening habits of a user can be useful to session-based video recommendations to her, and viceversa.

Finally, we advise the need for a more extensive evaluation of sequence-aware recommenders that goes beyond traditional recommendation accuracy. For example, little research has been conducted so far in the study of other aspects such as the novelty and diversity in sequence-aware recommender systems [137].

Bibliography

[1] *RecSys Challenge '16: Proceedings of the Recommender Systems Challenge*, New York, NY, USA, 2016. ACM.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 3–14, Washington, DC, USA, 1995. IEEE Computer Society.

[5] Natalie Aizenberg, Yehuda Koren, and Oren Somekh. Build your own music recommender by modeling internet radio streams. In *Proc. WWW*, pages 1–10, 2012.

[6] Vall Andreu, Markus Schedl, Widmer Gerhard, Massimo Quadrana, and Paolo Cremonesi. The importance of song context in music playlists: Enabling recommendations in the long tail. In *RecSys Posters*, 2017.

[7] Ricardo Baeza-Yates, Di Jiang, Fabrizio Silvestri, and Beverly Harrison. Predicting the next app that you are going to use. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, pages 285–294, New York, NY, USA, 2015. ACM.

[8] L. Baltrunas and X. Amatriain. Towards time-dependant recommendation based on implicit feedback. In *CARS'09: Workshop on Context-aware Recommender Systems*, pages 1–5, 2009.

[9] Chris Bateman, Rebecca Lowenhaupt, and Lennart E Nacke. Player typology in theory and practice. *Proceedings of DiGRA: Think Design Play*, 2011.

[10] Ron Begleiter, Ran El-Yaniv, and Golan Yona. On prediction using variable order markov models. *J. Artif. Int. Res.*, 22(1):385–421, December 2004.

[11] Robert M. Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proceedings of the 2007 Seventh IEEE International Con-*

ference on Data Mining, ICDM '07, pages 43–52, Washington, DC, USA, 2007. IEEE Computer Society.

[12] Alejandro Bellogin, Pablo Castells, and Ivan Cantador. Precision-oriented evaluation of recommender systems: An algorithmic comparison. In *RecSys'11: 5th ACM Conf. on Recommender Systems*, pages 333–336, 2011.

[13] Adam Berenzweig, Beth Logan, Daniel P. W. Ellis, and Brian P. W. Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Comput. Music J.*, 28(2):63–76, June 2004.

[14] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012.

[15] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proc. ISMIR*, pages 591–596. University of Miami, October 2011.

[16] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. *12th Int. Conf. on Music Information Retrieval (ISMIR)*, 2011.

[17] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[18] Geoffray Bonnin and Dietmar Jannach. Automated generation of music playlists: Survey and experiments. *ACM Comput. Surv.*, 47(2):26:1–26:35, November 2014.

[19] Robin Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

[20] Pedro G. Campos, Fernando Díez, and Iván Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1):67–119, 2014.

[21] Pedro G. Campos, Fernando Díez, and Iván Cantador. Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1-2):67–119, February 2014.

[22] Pedro G. Campos, Fernando Díez, and Manuel Sánchez-Montañés. Towards a more realistic evaluation: Testing the ability to predict future tastes of matrix factorization-based recommenders. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 309–312, New York, NY, USA, 2011. ACM.

[23] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[24] Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 714–722, New York, NY, USA, 2012. ACM.

[25] Shuo Chen, Jiexun Xu, and Thorsten Joachims. Multi-space probabilistic sequence modeling. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 865–873, New York, NY, USA, 2013. ACM.

[26] Chen Cheng, Haiqin Yang, Michael R. Lyu, and Irwin King. Where you like to go next: Successive point-of-interest recommendation. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 2605–2611. AAAI Press, 2013.

[27] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *SSST-8: 8th Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, 2014.

[28] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[29] Paolo Cremonesi, Franca Garzotto, Roberto Pagano, and Massimo Quadrana. Recommending without short head. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pages 245–246, New York, NY, USA, 2014. ACM.

[30] Paolo Cremonesi, Franca Garzotto, and Massimo Quadrana. Evaluating top-n recommendations "when the best are gone". In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 339–342, New York, NY, USA, 2013. ACM.

[31] Paolo Cremonesi, Franca Garzotto, and Roberto Turrin. *User-Centric vs. System-Centric Evaluation of Recommender Systems*, pages 334–351. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[32] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM.

[33] Paolo Cremonesi and Massimo Quadrana. Cross-domain recommendations without overlapping data: Myth or reality? In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 297–300, New York, NY, USA, 2014. ACM.

[34] Paolo Cremonesi and Roberto Turrin. Time-evolution of iptv recommender systems. In *Proceedings of the 8th European Conference on Interactive TV and Video*, EuroITV '10, pages 105–114, New York, NY, USA, 2010. ACM.

[35] Sally Jo Cunningham, David Bainbridge, and Annette Falconer. More of an art than a science: Supporting the creation of playlists and mixes. In *Proceedings of 7th International Conference on Music Information Retrieval*, pages 240–245, Victoria, Canada, 2006.

[36] Yann N Dauphin, Harm de Vries, Junyoung Chung, and Yoshua Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *CoRR*, 1502.04390, 2015.

[37] Yashar Deldjoo, Mehdi Elahi, Paolo Cremonesi, Franca Garzotto, Pietro Piazzolla, and Massimo Quadrana. Content-based video recommendation system based on stylistic visual features. *Journal on Data Semantics*, 5(2):99–113, 2016.

[38] Kristin Diehl, Erica van Herpen, and Cait Lamberton. Organizing products with complements versus substitutes: Effects on store preferences as a function of effort and assortment perceptions. *Journal of Retailing*, 91(1):1–18, 2015.

[39] N. Djuric, V. Radosavljevic, M. Grbovic, and N. Bhamidipati. Hidden conditional random fields with deep user embeddings for ad targeting. In *2014 IEEE International Conference on Data Mining*, pages 779–784, Dec 2014.

[40] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. DeCAF: A deep convolutional activation feature for generic visual recognition. In *ICML'14: 31st Int. Conf. on Machine Learning*, pages 647–655, 2014.

[41] Paul Dourish. What we talk about when we talk about context. *Personal Ubiquitous Comput.*, 8(1):19–30, February 2004.

[42] Sara Drenner, Shilad Sen, and Loren Terveen. Crafting the initial user experience to achieve community goals. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 187–194, New York, NY, USA, 2008. ACM.

[43] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. The yahoo! music dataset and kdd-cup'11. In *Proceedings of KDDCup 2011*, 2011.

[44] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[45] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. Personalized ranking metric embedding for next new poi recommendation. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 2069–2075. AAAI Press, 2015.

[46] Gernot A Fink. *Markov models for pattern recognition: from theory to applications*. Springer Science & Business Media, 2014.

[47] Arthur Flexer, Dominik Schnitzer, Martin Gasser, and Gerhard Widmer. Playlist generation using start and end songs. In *Proc. ISMIR*, pages 173–178, 2008.

[48] Florent Garcin, Christos Dimitrakakis, and Boi Faltings. Personalized news recommendation with context trees. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 105–112, New York, NY, USA, 2013. ACM.

[49] Zoubin Ghahramani. Hidden markov models. chapter An Introduction to Hidden Markov Models and Bayesian Networks, pages 9–42. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2002.

[50] Nadav Golbandi, Yehuda Koren, and Ronny Lempel. On bootstrapping recommender systems. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 1805–1808, New York, NY, USA, 2010. ACM.

[51] Carlos A. Gomez-Uribe and Neil Hunt. The Netflix recommender system: Algorithms, business value, and innovation. *Transactions on Management Information Systems*, 6(4):13:1–13:19, 2015.

[52] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[53] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1809–1818, New York, NY, USA, 2015. ACM.

[54] Negar Hariri, Bamshad Mobasher, and Robin Burke. Context-aware music recommendation based on latenttopic sequential patterns. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 131–138, New York, NY, USA, 2012. ACM.

[55] David Hauger, Markus Schedl, Andrej Košir, and Marko Tkalčič. The million musical tweets dataset: What can we learn from microblogs. In *Proceedings of the 14th International Society for Music Information Retrieval Conference (ISMIR 2013)*, Curitiba, Brazil, November 2013.

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[57] Qi He, Daxin Jiang, Zhen Liao, Steven C. H. Hoi, Kuiyu Chang, Ee-Peng Lim, and Hang Li. Web query recommendation via sequential query prediction. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 1443–1454, Washington, DC, USA, 2009. IEEE Computer Society.

[58] Ruining He and Julian McAuley. VBPR: Visual Bayesian Personalized Ranking from implicit feedback. *CoRR*, 1510.01784, 2015.

[59] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

[60] B. Hidasi and D. Tikk. Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback. In *ECML-PKDD'12, Part II*, number 7524 in LNCS, pages 67–82. Springer, 2012.

[61] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *International Conference on Learning Representations*, 2016.

[62] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 241–248, New York, NY, USA, 2016. ACM.

[63] Balázs Hidasi and Domonkos Tikk. General factorization framework for context-aware recommendations. *Data Mining and Knowledge Discovery*, 30(2):342–371, 2015.

[64] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[65] Mehdi Hosseinzadeh Aghdam, Negar Hariri, Bamshad Mobasher, and Robin Burke. Adapting recommendations to contextual changes using hierarchical hidden markov models. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 241–244, New York, NY, USA, 2015. ACM.

[66] Dietmar Jannach, Lukas Lerche, and Michael Jugovac. Adaptation and evaluation of recommendations for short-term shopping goals. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 211–218, New York, NY, USA, 2015. ACM.

[67] Dietmar Jannach, Lukas Lerche, and Iman Kamehkhosh. Beyond "hitting the hits": Generating coherent music playlist continuations with the right tracks. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 187–194, New York, NY, USA, 2015. ACM.

[68] Dietmar Jannach and Malte Ludewig. Determining characteristics of successful recommendations from log data - a case study. In *Proc. SAC '17*, 2017.

[69] Andreas Jansson, Colin Raffel, and Tillman Weyde. This is my jam Data dump. In *Proc. ISMIR*, page 175, 2015.

[70] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *MM'14: 22nd ACM Int. Conf. on Multimedia*, pages 675–678, 2014.

[71] Peter Knees, Tim Pohle, Markus Schedl, and Gerhard Widmer. Combining audio-based similarity with web-based data to accelerate automatic music playlist generation. In *Proc. International Workshop on Multimedia IR*, pages 147–154, 2006.

[72] Noam Koenigstein and Yehuda Koren. Towards scalable and accurate item-oriented recommendations. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 419–422, New York, NY, USA, 2013. ACM.

[73] Yehuda Koren. Collaborative filtering with temporal dynamics. *Commun. ACM*, 53(4):89–97, April 2010.

[74] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *NISP'12: 26th Annual Conf. on Neural Information Processing Systems 2012*, pages 1106–1114, 2012.

[75] Lukas Lerche, Dietmar Jannach, and Malte Ludewig. On the value of reminders within e-commerce recommendations. In *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*, UMAP '16, pages 27–35, New York, NY, USA, 2016. ACM.

[76] Benjamin Letham, Cynthia Rudin, and David Madigan. Sequential event prediction. *Mach. Learn.*, 93(2-3):357–380, November 2013.

[77] Defu Lian, Vincent W. Zheng, and Xing Xie. Collaborative filtering meets next check-in location prediction. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13 Companion, pages 231–232, New York, NY, USA, 2013. ACM.

[78] Kwan Hui Lim, Jeffrey Chan, Christopher Leckie, and Shanika Karunasekera. Personalized tour recommendation based on user interests and points of interest visit durations. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1778–1784. AAAI Press, 2015.

[79] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[80] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *CoRR*, 1506.00019, 2015.

[81] Beth Logan. Content-based playlist generation: Exploratory experiments. In *Proc. ISMIR*, 2002.

[82] P. Lops, M. Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, pages 73–105. Springer, 2011.

[83] Pasquale Lops, Marco de Gemmis, Giovanni Semeraro, Cataldo Musto, and Fedelucio Narducci. Content-based and collaborative techniques for tag recommendation: an empirical evaluation. *Journal of Intelligent Information Systems*, 40(1):41–61, 2013.

[84] E. H. C. Lu, Y. W. Lin, and J. B. Ciou. Mining mobile application sequential patterns for usage prediction. In *Granular Computing (GrC), 2014 IEEE International Conference on*, pages 185–190, Oct 2014.

[85] Nizar R. Mabroukeh and C. I. Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Comput. Surv.*, 43(1):3:1–3:41, December 2010.

[86] FranÃ§ois Maillet, Douglas Eck, Guillaume Desjardins, and Paul Lamere. Steerable playlist generation by learning song similarity from radio station playlists. In *In Proceedings of the 10th International Conference on Music Information Retrieval*, 2009.

[87] Christopher D. Manning and Hinrich SchÃ$\frac{1}{4}$tze. *Foundations of Statistical Natural Language Processing*. MIT Press, 2000.

[88] Benjamin M. Marlin and Richard S. Zemel. Collaborative prediction and ranking with non-random missing data. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 5–12, New York, NY, USA, 2009. ACM.

[89] J. J. McAuley, C. Targett, Q. Shi, and A. van den Hengel. Image-based recommendations on styles and substitutes. In *SIGIR*, 2015.

[90] Brian Mcfee and Gert Lanckriet. The natural language of playlists. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, pages 537–541, Miami (Florida), USA, October 24-28 2011. `http://ismir2011.ismir.net/papers/PS4-11.pdf`.

[91] Brian McFee and Gert RG Lanckriet. Hypergraph models of playlist dialects. In *Proc. ISMIR*, pages 343–348, 2012.

[92] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, 1310.4546, 2013.

[93] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[94] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH'10: 11th Ann. Conf. of the Int. Speech Communication Association*, pages 1045–1048, 2010.

[95] B. Mobasher, Honghua Dai, Tao Luo, and M. Nakagawa. Using sequential and non-sequential patterns in predictive web usage mining tasks. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 669–672, 2002.

[96] Omar Moling, Linas Baltrunas, and Francesco Ricci. Optimal radio channel recommendations with explicit and implicit feedback. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 75–82, New York, NY, USA, 2012. ACM.

[97] Joshua Moore, Shuo Chen, Douglas Turnbull, and Thorsten Joachims. Taste over time: the temporal dynamics of user preferences. In *Proceedings of the 14th International Society for Music Information Retrieval Conference*, November 4-8 2013. `http://www.ppgia.pucpr.br/ismir2013/wp-content/uploads/2013/09/220_Paper.pdf`.

[98] Miki Nakagawa and Bamshad Mobasher. Impact of site characteristics on recommendation models based on association rules and sequential patterns. In *Proceedings of the IJCAI*, volume 3, 2003.

[99] Nagarajan Natarajan, Donghyuk Shin, and Inderjit S. Dhillon. Which app will you use next?: Collaborative filtering with interactional context. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 201–208, New York, NY, USA, 2013. ACM.

[100] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 1310–1318, 2013.

[101] Steffen Pauws, Wim Verhaegh, and Mark Vossen. Fast generation of optimal music playlists using local search. In *Proceedings of the 7th International Conference on Music Information Retrieval*, Victoria (BC), Canada, October 8-12 2006. `http://ismir2006.ismir.net/PAPERS/ISMIR0631_Paper.pdf`.

[102] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[103] I. Pilászy and D. Tikk. Recommending new movies: Even a few ratings are more valuable than metadata. In *Recsys'09: ACM Conf. on Recommender Systems*, pages 93–100, 2009.

[104] John C. Platt, Christopher JC Burges, Steven Swenson, Christopher Weare, and Alice Zheng. Learning a Gaussian Process Prior for Automatically Generating Music Playlists. In *NIPS*, pages 1425–1432, 2001.

[105] Tim Pohle, Elias Pampalk, and Gerhard Widmer. Generating similarity-based playlists using traveling salesman algorithms. In *Proc. DAFx*, pages 220–225, 2005.

[106] Stephen R. Porter and Michael E. Whitcomb. The impact of lottery incentives on student survey response rates. *Research in Higher Education*, 44(4):389–407, 2003.

[107] Bruno Pradel, Nicolas Usunier, and Patrick Gallinari. Ranking with non-random missing ratings: Influence of popularity and positivity on evaluation metrics. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 147–154, New York, NY, USA, 2012. ACM.

[108] Pearl Pu, Li Chen, and Rong Hu. A user-centric evaluation framework for recommender systems. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 157–164, New York, NY, USA, 2011. ACM.

[109] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendaitons with hierarchical recurrent neural networks. In *Proceedings of the 11th ACM Conference on Recommender Systems*, RecSys '17, 2017.

[110] Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 273–282, New York, NY, USA, 2014. ACM.

[111] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press.

[112] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 811–820, New York, NY, USA, 2010. ACM.

[113] F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35. Springer US, 2011.

[114] Cynthia Rudin, Benjamin Letham, Ansaf Salleb-Aouissi, Eugene Kogan, and David Madigan. Sequential event prediction with association rules. In *COLT*, pages 615–634, 2011.

[115] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Internal Representations by Error Propagation, pages 673–695. MIT Press, Cambridge, MA, USA, 1988.

[116] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *Int. Journal of Computer Vision*, 115(3):211–252, 2015.

[117] Alan Said, Babak Loni, Roberto Turrin, and Andreas Lommatzsch. An extended data model format for composite recommendation. In *Proc. of the 8th RecSys conference 2014*, Foster City, Silicon Valley, CA, USA, 2014.

[118] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, August 1988.

[119] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW'01: Int. Conf. on World Wide Web*, pages 285–295, 2001.

[120] Martin Saveski and Amin Mantrach. Item cold-start recommendations: Learning local collective embeddings. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 89–96, New York, NY, USA, 2014. ACM.

[121] Markus Schedl. *Leveraging Microblogs for Spatiotemporal Music Information Retrieval*, pages 796–799. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[122] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. Recommendations as treatments: Debiasing learning and evaluation. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1670–1679. JMLR.org, 2016.

[123] Guy Shani, David Heckerman, and Ronen I. Brafman. An mdp-based recommender system. *J. Mach. Learn. Res.*, 6:1265–1295, December 2005.

[124] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *CVPR'14: IEEE Conf. on Computer Vision and Pattern Recognition Workshops*, June 2014.

[125] Márcio Soares and Paula Viana. Tuning metadata for better movie content-based recommendation systems. *Multimedia Tools Appl.*, 74(17):7015–7036, September 2015.

[126] Qiang Song, Jian Cheng, Ting Yuan, and Hanqing Lu. Personalized recommendation meets your next favorite. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 1775–1778, New York, NY, USA, 2015. ACM.

[127] Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 553–562, New York, NY, USA, 2015. ACM.

[128] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[129] Harald Steck. Training and testing of recommender systems on data missing not at random. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 713–722, New York, NY, USA, 2010. ACM.

[130] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013.

[131] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, 1409.4842, 2014.

[132] Yukihiro Tagami, Hayato Kobayashi, Shingo Ono, and Akira Tajima. Modeling user activities on the web using paragraph vector. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 125–126, New York, NY, USA, 2015. ACM.

[133] Maryam Tavakol and Ulf Brefeld. Factored mdps for detecting topics of user sessions. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 33–40, New York, NY, USA, 2014. ACM.

[134] Ignacio Fernández Tobías. *A semantic-based framework for building cross-domain networks: Application to item recommendation*. Ph.d. thesis, 2013.

[135] Bartlomiej Twardowski. Modelling contextual information in session-aware recommender systems with neural networks. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 273–276, New York, NY, USA, 2016. ACM.

[136] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in Neural Information Processing Systems*, pages 2643–2651, 2013.

[137] Saúl Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 109–116, New York, NY, USA, 2011. ACM.

[138] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *KDD'15: 21th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 1235–1244, 2015.

[139] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. Learning hierarchical representation model for nextbasket recommendation. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 403–412, New York, NY, USA, 2015. ACM.

[140] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, June 1989.

[141] Qiang Wu, Christopher J. Burges, Krysta M. Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. *Inf. Retr.*, 13(3):254–270, June 2010.

[142] Xiang Wu, Qi Liu, Enhong Chen, Liang He, Jingsong Lv, Can Cao, and Guoping Hu. Personalized next-song recommendation in online karaokes. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 137–140, New York, NY, USA, 2013. ACM.

[143] Liang Xiang, Quan Yuan, Shiwan Zhao, Li Chen, Xiatian Zhang, Qing Yang, and Jimeng Sun. Temporal recommendation on graphs via long- and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 723–732, New York, NY, USA, 2010. ACM.

[144] J. Xu, T. Xing, and M. van der Schaar. Personalized course sequence recommendations. *IEEE Transactions on Signal Processing*, 64(20):5340–5352, Oct 2016.

[145] Ghim-Eng Yap, Xiao-Li Li, and Philip S. Yu. Effective next-items recommendation via personalized sequential pattern mining. In *Proceedings of the 17th International Conference on Database Systems for Advanced Applications - Volume Part II*, DASFAA'12, pages 48–64, Berlin, Heidelberg, 2012. Springer-Verlag.

[146] Hong Yu and Mark O. Riedl. A sequential recommendation approach for interactive personalized story generation. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '12, pages 71–78, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

[147] Eva Zangerle, Martin Pichl, Wolfgang Gassler, and Günther Specht. #nowplaying music dataset: Extracting listening behavior from twitter. In *Proceedings of the First International Workshop on Internet-Scale Multimedia Management*, WISMM '14, pages 21–26, New York, NY, USA, 2014. ACM.

[148] Yuyu Zhang, Hanjun Dai, Chang Xu, Jun Feng, Taifeng Wang, Jiang Bian, Bin Wang, and Tie-Yan Liu. Sequential click prediction for sponsored search with recurrent neural networks. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pages 1369–1375. AAAI Press, 2014.

[149] Gang Zhao, Mong Li Lee, Wynne Hsu, and Wei Chen. Increasing temporal diversity with purchase intervals. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 165–174, New York, NY, USA, 2012. ACM.

[150] Gang Zhao, Mong LI Lee, and Hsu Wynne. Utilizing purchase intervals in latent clusters for product recommendation. In *Proceedings of the 8th Workshop on Social Network Mining and Analysis*, SNAKDD'14, pages 4:1–4:9, New York, NY, USA, 2014. ACM.

[151] B. Zhou, S. C. Hui, and K. Chang. An intelligent recommender system using sequential web access patterns. In *Cybernetics and Intelligent Systems, 2004 IEEE Conference on*, volume 1, pages 393–398 vol.1, Dec 2004.

[152] Andrew Zimdars, David Maxwell Chickering, and Christopher Meek. Using temporal data for making recommendations. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, UAI '01, pages 580–588, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.