Heterogeneous User Actions in Recommender Systems

Tural Gurbanov

Supervisor: Prof. Francesco Ricci

Faculty of Computer Science
Free University of Bozen-Bolzano

This dissertation is submitted for the degree of
Doctor of Philosophy

January 2020

Acknowledgements

First and foremost, the author would like to express sincere gratitude to Prof. Francesco Ricci for his patience, immense knowledge, and continuous support for the entirety of this Ph.D. study and research. Thanks are also in order to the esteemed reviewers, Prof. Dipl. Ing. Dr. Dietmar Jannach and Dr. Alexandros Karatzoglou, for their valuable comments, which allowed this thesis to reach its full potential.

Besides that, it is important to acknowledge SoundCloud[1] for supporting the author's internship and for backing our initiatives in the area of recurrent neural networks. We would also like to thank *ivi*[2] for providing us with their valuable data and helping us with both offline and online experiments. We thank the engineers of *ivi*, particularly Maria Tarakanova and Alexander Andrusenko, who helped to implement several multi action prediction models and to incorporate them into the *ivi* ecosystem. Additionally, we would like to acknowledge Timur Bagautdinov, Alexey Rodriguez Yakushev, David Massimo and other *unibz* colleagues for many helpful discussions. Finally, the author would like to thank Maxim Ionov for stimulating his creativity.

[1] https://soundcloud.com/
[2] https://www.ivi.tv/

Abstract

Recommender Systems (RSs) are software tools and techniques providing to users suggestions for items, such as what movies to watch, what music to listen to, or what items to buy. These suggestions are usually personalized, i.e., they are adapted to the user's known preferences, which are either explicitly expressed, for example, in the form of ratings for items, or are inferred by interpreting online user activity. In many real-world scenarios obtaining preference information explicitly can be either difficult or even impossible. Hence, increasingly more RSs are built by leveraging abundant *implicit feedback* data, such as the log of user actions (e.g., clicks, views, or purchases), which only indirectly signal users' preferences or opinions.

Recently, various RS techniques employing implicit feedback datasets have emerged. They implicitly assume that if a user performs a domain dependent *target action* on an item, then she likes it. Therefore, implicit feedback models are designed to predict on which items the user will perform a target action. These items are then used to create recommendations.

To make these predictions, the majority of existing models consider only actions of a single type, such as video views. However, the interactions between a user and an item are rarely limited to actions of a single type. Thus, the single action models, while providing efficient results, ignore information that could be extracted from other types of actions, for instance, clicks or bookmarks.

The information about actions of multiple types can be jointly used to build more comprehensive user models that will better describe users' preferences and as a consequence will allow creating better RSs. The use of this type of information in RSs requires addressing challenges related to the identification of correlations between action types, and the elicitation of predictive actions for a target action type. Although some preliminary works have been done in this direction, a general model exploiting a combination of action of various types has not been created yet.

This *thesis* explores some challenges and questions that have not been addressed in previous research. It starts with an overview of the historical categorization of implicit feedback types and the current state of research on single and multi action prediction models for RSs. After that, different multi action prediction models, designed by the author of the

thesis, are presented in the order of increasing complexity. The first models, which are based on implicit matrix factorization, exploit correlations between different action types through heuristics determined by exploratory analysis of the data. The heuristics are usually domain dependent and may overlook subtle relations between actions of different types. This makes models less flexible and difficult to reuse. For this reason, we have then analyzed models that use machine learning and sequence mining techniques to find correlations between different action types automatically. The automation makes the process of finding correlations between action types easier and more precise, which in turn increases the prediction accuracy of the models. It also allows considering information about the ordering of the actions and the time delays between them. Finally, more sophisticated deep learning based models have been examined. These models take into account not only the sequence of actions performed by users, but also additional information, such as an action context and user and item features.

The comprehensive empirical evaluation, which was conducted on large real-world datasets, shows that using multiple actions is beneficial and it can outperform state-of-the-art single-type implicit feedback models. The evaluation also shows that models which can be used out-of-the-box in different domains and which utilize multiple action types and contextual information are superior to the other models that we have studied. These findings are also confirmed with online experiments. Finally, the analysis of models' predictions helps to explain the peculiarities of the models and provide hints about when and which models is better to use.

Table of contents

List of figures **xi**

List of tables **xiii**

1 Introduction **1**

1.1 Introduction and Motivation 1
1.2 Problem Statement 3
1.3 Contributions 4
1.4 Outline 6

2 Related Works **7**

2.1 Implicit Feedback 7
2.1.1 Categorization of User Actions 8
2.1.2 Challenges of Using Implicit Feedback 9
2.2 Action Prediction Models 11
2.2.1 Basic Collaborative Filtering Models 12
2.2.2 Sequence-aware Models 18
2.2.3 Hybrid Models 22
2.2.4 Multi Action Type Prediction Models 23
2.3 Conclusions 25

3 General Action Prediction Model **27**

3.1 General Model and Examples 27
3.1.1 Predicting Video Views with Open Details Actions 28
3.1.2 Predicting TV show Views with Watch Later Actions 30
3.2 7TV Dataset 31
3.3 Evaluation 31
3.4 Conclusions 33

4 Automated Multi Action Type Matrix Factorization **35**
4.1 Hybrid Prediction Model . . . 35
4.2 Predicting a Target Action with a User-item Interaction History . . . 38
4.2.1 History-summarizing Approaches . . . 39
4.2.2 Sequence-aware Approaches . . . 39
4.3 Datasets . . . 42
4.3.1 XING Dataset . . . 42
4.3.2 Rekko Dataset . . . 43
4.3.3 Taobao Dataset . . . 44
4.4 Evaluation Scenarios . . . 45
4.4.1 Predicting Replies with Click and Bookmark Actions . . . 45
4.4.2 Predicting Replies with All the Available Information . . . 48
4.4.3 Comparing Multi Action Prediction Models . . . 50
4.5 Evaluation . . . 51
4.5.1 Training and Testing Sets . . . 51
4.5.2 Protocol and Metrics . . . 52
4.5.3 Comparing Single and Multi Action Prediction Models . . . 53
4.5.4 Comparing History-summarizing and Sequence-aware Approaches for AMMF . . . 55
4.5.5 Comparing Multi Action Prediction Models . . . 57
4.6 AMMF in Action . . . 59
4.6.1 Introduction to *ivi* . . . 59
4.6.2 Prediction Models . . . 62
4.6.3 Offline Evaluation . . . 64
4.6.4 Results . . . 66
4.7 Conclusions . . . 67

5 Personalized Ranking with Long- and Short-term User Preferences **69**
5.1 Action Prediction Based on User Interaction Histories . . . 70
5.2 Action Prediction Model Architecture . . . 72
5.2.1 Predicting Target Actions Using User and Action Embeddings . . . 73
5.2.2 Ignoring Contextual Information While Predicting Target Actions . . . 74
5.2.3 Predicting Target Actions Using Only Recent User Actions . . . 75
5.2.4 Predicting Target Actions Using Only Actions of the Target Type . . . 76
5.3 Training Procedure . . . 77
5.4 Evaluation . . . 80
5.4.1 Comparing LSTP Variations . . . 80

5.4.2 Comparing RNN-based Multi Action Prediction Models 85
5.5 Conclusions 86

6 Conclusions **89**
6.1 Summary 90
6.2 Future Directions 91

Bibliography **93**

Appendix A Publications **103**
A.1 Modeling and Predicting User Actions in Recommender Systems 103
A.2 Action Prediction Models for Recommender Systems Based on Collaborative Filtering and Sequence Mining Hybridization 104
A.3 Context-Aware Integrated Development Environment Command Recommender Systems 104
A.4 Exploiting Multiple Action Types in Recommender Systems 104
A.5 A Graphical User Interface for Presenting Integrated Development Environment Command Recommendations: Design, Evaluation, and Implementation 105
A.6 Improving Integrated Development Environment Commands Knowledge with Recommender Systems 105
A.7 User Preference Elicitation, Rating Sparsity and Cold Start 106

List of figures

2.1 Examples of user-based (left) and item-based (right) CF approaches 13
2.2 Examples of indicator, observation, and confidence matrices 15
2.3 Graphical representation of the bagging technique for the target action prediction task 16

3.1 The distribution of probability that a user will start to watch a video after Δ days since the video details page was opened 29

4.1 The prediction pipeline of the AMMF model 36
4.2 The distribution of the time delays between actions of all users. The largest delay is 400 seconds. 49
4.3 The popularity, recency, and uniqueness of the top-k items recommended by the models 56
4.4 The main page of the *ivi* application for Android-based smartphones. The page consists of blocks, such as Central Promo Block (CPB) or item collections. 61
4.5 The main pages of *ivi* application for smartphones (left) and smartTVs (right). 66

5.1 Graphical representation of the hierarchical RNN model which consists of the session- and user-level RNNs 70
5.2 Graphical representation of the LSTP model inference. The model combines the recent user activity with the long-term user preferences through an RNN, to predict future user actions. 72
5.3 The toy interaction history h_e which contains observations of clicks and replies performed by the user e on various items 73
5.4 The LSTP model architecture 74
5.5 Three modifications of the LSTP architecture (only the input part): aLSTP (left), tLSTP (center), iLSTP (right) 75
5.6 The rLSTP model which takes into account only actions of the target type . 76
5.7 Data used for the training of the LSTP model and its variations 77

5.8 Different approaches for an RNN training: (a) seq-to-seq, (b) seq-to-seq conditioned on action types, (c) seq-to-vec, where i_t denotes the item on which the target action type was performed 78
5.9 Generating a set of training prefixes from a user interaction history 79
5.10 The optimal architecture of the LSTP model for the evaluation on the XING dataset 81
5.11 The popularity and recency of the top-k recommendations provided by the models 84

List of tables

1.1 Crucial features of the action prediction models presented in the thesis . . . 5

2.1 User action types employed in implicit feedback models 8

3.1 Statistics of the used datasets 31
3.2 "Video view stopped action" record 31
3.3 Comparison of WR-MF and MMF ($f = 20$) 33

4.1 Crucial features of AMMF and its constituting prediction models 36
4.2 A sample of the interactions data 42
4.3 Statistics of the XING data 43
4.4 Statistics of the Rekko data 44
4.5 Statistics of the Taobao data 44
4.6 Action types sorted in ascending order of the preference level 51
4.7 Parameter values on which the search was carried out 53
4.8 The models' hyper-parameters and their optimal values 54
4.9 Comparison of models in terms of MAP@k and F1@k. Best values are underlined. 54
4.10 Statistics of the indicator matrices 55
4.11 The models' hyper-parameters and their optimal values 55
4.12 Comparison of models in terms of MAP@k and F1@k. Best values are underlined. 56
4.13 AMMF(LR-nt) and MF-BPR parameter values on which the search was taken. The "Common" row contains hyper-parameters related to both models. 57
4.14 The AMMF(LR-nt) and MF-BPR models' hyper-parameters and their optimal values for the XING, Rekko, and Taobao datasets 57
4.15 The comparison of AMMF(LR-nt) and MF-BPR in terms of MAP@k and F1@k on the XING, Rekko, and Taobao datasets. Best values are underlined. 58

4.16 The statistical analysis of the XING, Rekko, and Taobao datasets for the comparison of the AMMF(LR-nt) and MF-BPR models 59
4.17 The hyper-parameters' values for grid search 65
4.18 The models' hyper-parameters and their optimal values 65
4.19 Comparison of models in terms of MAP@k and F1@k. Best values are underlined. 66
4.20 The changes in business metrics observed in the test group of the AB-test. . 67

5.1 The features of the LSTP model and its variations. 73
5.2 The MF-based models' hyper-parameters and their values 81
5.3 Parameter values for the grid search optimization of the LSTP variations . . 82
5.4 The optimal values for the LSTP variations' hyper-parameters 82
5.5 Comparison of models in terms of MAP@k and F1@k. Best values are underlined. Values in bold highlight the cases when the performance difference between LSTP and the next best model is statistically significant with $p < 0.05$ (one-sided t-test). 83
5.6 The comparison of the STP models with fixed and trainable item embeddings 83
5.7 Parameter values for the grid search optimization of the CRNN model . . . 85
5.8 The RNN-based models' hyper-parameters and their optimal values for the XING, Rekko, and Taobao datasets 85
5.9 The comparison of LSTP and CRNN in terms of MAP@k and F1@k on the XING, Rekko, and Taobao datasets. Best values are underlined. 86

Chapter 1

Introduction

1.1 Introduction and Motivation

Personalization is the process of tailoring products and services to individual users' characteristics or preferences. It has become an integral part of our digital life which influences decisions we make and content we access. Movies on Netflix, items on Amazon, music on Spotify, and news on Google News are automatically personalized based on our behavior traces (e.g., video views and item purchases) and preferences, such as ratings and likes [87]. This kind of personalization is done by means of Recommender Systems (RSs) which are software tools and techniques providing personalized suggestions for items that are predicted to be of interest to the user [88].

RSs are information processing systems that actively gather various kinds of data to build recommendations. Data is primarily about the items to suggest and the users who will receive these recommendations [88]. To provide personalized item suggestions, users' preferences have to be known by the RS. These preferences are collected from feedback provided by users on items. The feedback can be either *explicit* or *implicit*. Unlike explicit feedback, where a user provides her preferences in the form of ratings or explicit likes, when implicit feedback is used, the system must infer the user's opinion from the user's action (clicks, purchases, views, etc.).

Earlier, research on RSs was mainly focusing on models utilizing explicit feedback. However, in many real-world scenarios this type of feedback is difficult to obtain or unavailable (e.g., news portals). For this reason, increasingly more recommenders are now built by leveraging (only) implicit feedback data. The advantage of implicit feedback is that it is available in many different domains, there is no need for users to invest time in expressing their preferences (no cognitive load on the user), and that potentially every user interaction with the system can be considered as a feedback [14]. At the same time, implicit

feedback has several characteristics which make its usage in RSs challenging. The most crucial among them is that implicit feedback can be ambiguous, i.e., it may not always be clear if the interaction of a user with an item should be interpreted as positive or negative feedback. Moreover, the interpretation of the same type of interactions may vary depending on a user and her previous interactions.

Because implicit feedback can be collected ubiquitously the potential impact of it might be even greater than that of explicit feedback [73]. For example, Stevens [97] observed that implicit feedback which operates constantly and non-intrusively was effective for tracking long-term user interests. Three sources of implicit evidence were used: whether a message was read or ignored, whether it was saved or deleted, and whether or not a follow up message was posted [73]. Morita et al. [69] showed a strong positive correlation between reading time and explicit feedback provided by users. They discovered that treating messages that the user read for more than 20 seconds as relevant produced better recall and precision in an information filtering simulation than would have been achieved using the messages explicitly rated by the user as relevant. Analogously, the results of the user study conducted by Konstan et al. [49] indicated that recommendations based on reading time could be nearly as accurate as recommendations based on explicit feedback.

The above mentioned findings have led to the emergence in recent years of various approaches employing implicit feedback datasets for RSs. These approaches assume the existence of user behavioral patterns that highly correlate with explicit user feedback. In particular, they implicitly assume that if a user performs a *target action* on an item, then she is interested in this item. Thus, a news article can be considered as a good recommendation for a user if the user read it.

Typically, implicit feedback models predict on which items the user will perform a target action. These items are then considered as good recommendations. To make predictions, implicit feedback models examine users' actions. Though the interactions between a user and an item are rarely limited to a single type of actions, the majority of existing models consider only actions of a single type, such as video views or purchases, when the recommender goal is to suggest novel interesting videos for the user. In general, there are two main reasons for this simplification. First of all, initial works on predictions employing implicit feedback were trying to reuse prediction models designed for explicit feedback. These models are mainly based on user ratings which are observations of a single type. Secondly, until recently there were no publicly available datasets containing user actions of multiple types. The appearance of such datasets revealed the possibility of improving a target action prediction by leveraging information about multiple action types (clicks, views, purchases, etc.) [32, 98, 101]. This

information can be jointly used to build more comprehensive user models which will better describe users' preferences and as a consequence will allow creating better RSs.

It is also true that not all available action types are relevant for the prediction of a target action. For instance, it is unlikely that the action "user purchased a book" depends on the action "user signed off from a book service". Generally, the elicitation of predictive action types for a target action type is done manually by using different heuristics. The set of heuristics and methods of elicitation usually depends on the domain. For example, Parra et al. [78] hypothesized a number of variables (i.e., types of actions) that may influence the prediction result. After that, they measured the effect of each variable and created a weighted linear model that employs these measurements. Analogously, heuristics are used to determine correlations between action types, which is a complex task requiring optimization of multiple hyper-parameters. That is why the exploration of methods that automatically discover predictive action types and correlation between them is important and can improve recommendation results.

1.2 Problem Statement

This *thesis* explores various challenges and open questions that have not been addressed in previous research. In particular, we study action prediction models employing user action data of multiple types which can be used to build more relevant item recommendations to the users. To this end, we seek to answer the following research questions.

RQ1: How to generate relevant recommendations by exploiting information about the full variety of actions that a user may perform while interacting with the system?

It is still an open, though important, question how to leverage users' actions to generate relevant recommendations. By analyzing users' actions, which are only traces of users' behavior, it is challenging to understand users' real preferences and opinions. Thus, the problem of finding the relationship between users' actions and preferences is crucial and should be taken into account.

RQ2: How to discover automatically dependency relations between action types?

To use multiple action types in an action prediction model one needs to establish their mutual dependency. However, the growth of the number of action types makes it difficult to explore manually which action types can be successfully used to predict a target action. For example, not all the available action types may be relevant for the prediction of a specific target action, while certain action types may be informative only

in combination with others. The exploration of methods that automatically identify the predictive action types and their dependency relations is essential and can improve recommendation results.

RQ3: How to exploit the temporal patterns and information about the ordering of the users' performed actions?

A user usually interacts with a service through a sequence of actions of different types. Previous works have shown that the use of temporal patterns and information about the ordering of actions can improve the accuracy of target action prediction models. However, most of existing sequence-aware models employ only actions of a single type [82]. By incorporating additional information available in other types of user actions into sequence-aware prediction models, one can further improve the recommendation results.

Thus, this thesis addresses the tasks of building domain-independent multi action prediction models for RSs that can automatically elicit predictive action types for a pre-defined target action type, determine correlations between action types, and leverage temporal patterns and information about the ordering of actions.

1.3 Contributions

The results presented in this thesis can be split into two groups. The first group, which forms the core of Chapters 3 and 4, contains results that have been already published in the following papers.

1. Gurbanov, T., Ricci, F., and Ploner, M. (2016). Modeling and predicting user actions in recommender systems. In *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*, UMAP '16, pages 151–155, New York, NY, USA. ACM

2. Gurbanov, T. and Ricci, F. (2017a). Action prediction models for recommender systems based on collaborative filtering and sequence mining hybridization. In *Proceedings of the Symposium on Applied Computing*, SAC '17, pages 1655–1661, New York, NY, USA. ACM

The first paper introduces a general multi action prediction model (MMF - Multi action type Matrix Factorization) for implicit feedback RSs. MMF predicts a target user action by leveraging information about actions of multiple types. The model exploits correlations

between different action types through heuristics determined by exploratory analysis of the data.

The second document introduces the results of research related to the use of sequentially-ordered actions of several types enriched with time information. The models presented in the paper have been used as a basis for creating an improved model called AMMF (Automated Multi action type Matrix Factorization) and a novel sequence-aware action prediction model called LSTP (Long- and Short-Term Preference). Both AMMF and LSTP have not been published yet and are introduced for the first time in Chapters 4 and 5. AMMF and LSTP are domain-independent multi action prediction models for RSs which can automatically determine correlations between action types and leverage temporal patterns and information about the ordering of actions. These models, as well as their analysis, form the second group of the results presented in the thesis.

The author of this thesis has also worked on a novel application of action prediction models, namely, for building recommendations for Integrated Development Environments (IDEs). In particular, the author contributed to the design, implementation, offline and online evaluation of the context-aware IDE command RS called CNTX [22–24]. The CNTX algorithm provides command recommendations by taking into account the contexts in which a particular user is observed and the contexts in which different command actions are usually executed. The proposed algorithm suggests to users novel commands, which are not likely to be discovered without the help of the recommender. However, since the discussions on IDE command RSs go beyond the specific topic of multi action prediction models, we decided not to present CNTX in detail in this thesis.

Thus, the main contributions of this thesis are three action prediction models for implicit feedback RSs that leverage additional contextual information and information available in actions of multiple types (Table 1.1).

Table 1.1: Crucial features of the action prediction models presented in the thesis

	MMF	AMMF	LSTP
Employs actions of multiple types	+	+	+
Finds correlations between actions automatically		+	+
Distinguishes between long- and short-term user preferences			+
Employs time and ordering of actions		+	+
Can incorporate contextual, user- and item-related information			+

The conducted offline and online studies show a significant improvement in the target action prediction accuracy when actions of multiple types are employed. Particularly, the

presented models can be used to recommend to a user less popular but at the same time relevant items by employing information available in user actions of various types and by taking into account short- and long-term user preferences. Thus, it has been shown that it is possible to create more effective RSs that will not require users to specify their preferences explicitly and will use massive and presumably information-rich data generated by user activity. Such systems can be then utilized in areas where users do not or cannot provide only explicit feedback to the system (e.g., news portals, streaming music and video services, tourism applications).

The source code of all the models and experiments presented in this thesis is available at https://gitlab.inf.unibz.it/tural-gurbanov/mapm. A complete list of the author's publications supported with brief descriptions is presented in Appendix A.

1.4 Outline

The rest of the manuscript is organized as follows. Chapter 2 provides an overview of the historical categorization of implicit feedback types and the current state of research on action prediction models for RSs. In Chapter 3 we present a general multi action prediction model which is based on implicit matrix factorization. The model exploits correlations between different action types through heuristics determined by exploratory analysis of the data. An extension of the general prediction model leveraging machine learning techniques to find correlations between different action types automatically is presented in Chapter 4. The automation helps to increase the target action prediction accuracy and allows to take into account additional information, such as temporal delays between actions or the order of actions. Further improvements to the action prediction models are presented in Chapter 5 where a new sequence-aware prediction model employing actions of multiple types is considered. Apart from automatically discovering the relationships between actions types and exploiting the temporal patterns, the model also takes into account short-term user interests to predict whether a user will perform a target action on an item. Finally, in Chapter 6 we summarize our findings and conclude the manuscript with future directions of research.

Chapter 2

Related Works

2.1 Implicit Feedback

To provide recommendations an RS must have some knowledge of the user preferences, which in many approaches are inferred by analyzing the feedback of the user on items. The most obvious solution is to use explicit ratings, where users inform the system about what they think about some items (e.g., a movie). Movie ratings, hotel reviews, and restaurant "stars" are popular forms of explicit feedback. However, explicit feedback suffers from a drawback: there can be a discrepancy between what the users say and what they do [75]. Moreover, the cognitive effort to assign accurate ratings acts as a disincentive to enter them [72]. This is one of the reasons why only a small percentage of users express explicit user feedback, which in turn leads to the data sparsity problem [45]. Hence, explicit ratings, while common and trusted, may in fact not be fully reliable.

Implicit feedback techniques seek to avoid the drawbacks of explicit feedback mentioned above by inferring the user's opinion from the user's actions (clicks, purchases, views, etc.) that are collected and stored by the system [72]. Thus, potentially every user interaction with the system can be considered as a feedback [14]. However, implicit feedback data only indirectly signals users' preferences and opinions. Often a single feedback can only be interpreted with a certain degree of uncertainty: it is unclear whether the recorded interaction signals are positive, negative, or somewhere in between. For example, if a user purchased an item, it does not always mean that she liked it. Besides, even if an implicit feedback signal can be interpreted accurately, there is still the question of how to quantify and how to combine it with other types of implicit and explicit feedback.

Trying to understand the nature of implicit feedback better, researchers studied simple user actions (such as scrolling, clicking, and bookmarking) and their correlation with explicitly provided user interests (grades or ratings) [14, 49, 69]. These studies established a general

Table 2.1: User action types employed in implicit feedback models

		Scale		
		Segment	Object	Class
Category	Examine	Eye gaze View Listen	Select Click	
	Retain		Bookmark Save Delete Purchase	Subscribe
	Reference	Quote	Share Link Reply Cite	
	Annotate	Highlight		Organize

feedback types categorization framework and revealed the challenges related to the use of the implicit feedback data in RSs. Both topics are discussed below.

2.1.1 Categorization of User Actions

The first comprehensive analysis of implicit feedback with a focus on its use in information filtering systems was done by Nicholos [70]. He introduced a list of actions (observable behaviors), such as purchase, reply, repeated use and others, that can be used to acquire user preferences. Oard and Kim [72, 73] extended that work by presenting a framework that groups observable behaviors into categories. Later on, Jannach et al. [43] studied actions in terms of multiple domains and proposed another extension to that framework. The authors updated the list of observable behaviors and introduced two new categories, namely "Social & Public Actions" and "Physical Actions". Though the framework proposed by Jannach et al. [43] is more up-to-date, it is less structured and contains overlapping actions. That is why we use the framework proposed by Oard and Kim [72, 73] to present the user action types typically employed in implicit feedback models. For this reason, we removed the *rate* action from the original framework and revised the remaining action types to fit the more recent user/system interaction scenarios better (e.g., we excluded the *print* action as obsolete and added the *eye gaze* action). The updated framework is shown in Table 2.1.

The vertical dimension of the framework is the purpose of the actions which categorizes observable behaviors into four broad categories, namely *examine*, *retain*, *reference* and *annotate*. The *examine* category groups behaviors that can provide evidence regarding the

user's assessment of information objects. The *retain* category consists of actions suggesting some degree of intention to make future use of an object. Each activity in the *reference* category has the effect of establishing some form of link between two objects. Observable behaviors in the *annotate* category are actions that intentionally add value to an information object.

The horizontal dimension of the framework shows the most specific scope of the object being manipulated. It divides the observable behaviors into three levels (*segment*, *object* and *class*) based on the scale at which the observations were made. By "the most specific scope" we mean the smallest unit normally associated with the behavior – a behavior applied to the scope may be replicated at larger scopes, but not normally at smaller scopes. For example, a user may view a portion of a document, select an entire document or subscribe to a collection of documents. The scale values have been chosen to be suitable both for text and non-text (e.g., video or music) item descriptions.

The framework was designed to provide developers with a useful perspective on the broad range of observable behaviors that might be used as a basis for operating on information content. Each behavior is considered independently from others which makes it possible to assign it to a specific category. However, in the real-world applications the behaviors are observed in sequences where each new action may depend on the previous ones. Thus, the purpose of an observable behavior might be ambiguous.

2.1.2 Challenges of Using Implicit Feedback

Historically, research in the field of RSs was motivated by scenarios in which users express their preferences in the form of explicit item ratings [45]. This led to the development of algorithms that are able to predict accurately which rating a user would give to a certain item. To reuse these technologies, initial works on implicit feedback employed techniques transforming and encoding preference signals to explicit feedback. Standard recommender system algorithms for rating prediction were later applied to the transformed data. For example, Parra et al. [78] discovered a relation between the number of times users listen to an album, the time elapsed since the users interacted with the album, and the rating the users report. Based on this observation, the authors proposed a linear regression model that predicts a user rating, given information of how the user interacted with an item. The resulting ratings can be used as an input data for explicit feedback models.

The question of how to transform preference signals, expressed with actions, into explicit feedback can be challenging and is often solved in an arbitrary manner [41, 50]. Various studies have been conducted by researches to investigate the relationship between explicit

ratings and implicit feedback actions [14, 80, 109]. The obtained results vary depending on the domain and experimental setup and are hard to interpret.

Claypool et al. [14] conducted a laboratory study in which participants were asked to freely browse the Web for 20-30 minutes. Every time a participant was leaving a web page, she had to rate it with respect to how interesting its contents were. The recorded user actions, such as mouse movements and dwelling times, were then compared with the collected explicit ratings. The analysis revealed that the time spent and the scrolling activity on a web page correlates with explicit ratings. Other indicators, such as mouse movement and clicks, had no clear relation with the participant's interest.

A similar study on the relationship between various types of browsing actions and explicit interest statements was reported by Shapira et al. [90]. The strongest correlations with the explicit ratings were found for the indicator "time of mouse movement relative to reading time" and the number of visited links on the page. Note that mouse movements were not considered to be a good indicator according to the study of Claypool et al. [14] discussed above.

Zigoris et al.[109] showed that in the news domain, when both explicit and implicit feedback are available, the implicit feedback possesses only limited predictive value. Thus, the recommendations based on the combination of both feedback types was only marginally better than when using explicit feedback alone. On the other hand, Pizzato et al. [80] showed that in an online dating platform, recommendations built on implicit feedback can be more accurate than those built on explicit preference information. Such contradictions emphasize once more that the interpretation of implicit feedback can be highly dependent on the respective domain.

According to Hu et al. [38], implicit feedback has several crucial characteristics which prevent the direct use of recommendation algorithms that were designed for explicit feedback:

1. Usually there is no negative feedback.
   If a user did not act on an item that does not mean that the user does not like the item.

2. Implicit feedback is noisy.
   While we passively track the users' behavior, we can only guess their preferences and real motivations for the actions. For example, if the user purchases an item there is no clear indication of the user's preference for this item: the item might be purchased as a gift, or the user might be finally disappointed by the true experience/usage of the item.

3. Preference vs. Confidence.
   Systems based on explicit feedback let the user express her level of preference, e.g., by rating the item from 1 ("totally dislike") to 5 ("really like"). Hence, the numerical

value of explicit feedback indicates a preference. Instead, the numerical values of implicit feedback typically measure the frequency of the observed actions, e.g., how many time a user interacted with a certain item [55] or how much time the user watched a certain show [38]. A larger value does not indicate a higher preference. For example, the most loved show may be a movie that the user will watch only once, while there is a series that the user quite likes and watches every week [38]. However, the numerical value of the feedback may increase the *confidence* we have in an observation. A single event might be caused by various reasons that have nothing to do with user preferences. At the same time, a recurring event is more likely to signal a user opinion. Thus, the numerical value of implicit feedback indicates confidence on whether the user likes the item [38].

4. Evaluation of implicit feedback recommenders.
   The goal of an RS is to identify items that will be of interest to a certain user [17]. However, implicit feedback contains only information about user behavior, not preferences. This complicates not only the creation of an RS but also the evaluation process. The use of traditional prediction metrics such as mean squared error is inappropriate since the test data for implicit feedback models does not contain negative feedback/actions.

Therefore, more sophisticated techniques were proposed in the literature to deal with implicit feedback. These techniques instead of transforming user actions into explicit ratings consider them as separate signals. The approaches, such as one-class collaborative filtering [38, 76, 85], sequence mining techniques [35, 68], and hybrid models [10] are considered in the reminder of this section.

2.2 Action Prediction Models

To provide personalized suggestion for items to a user, the RS should at first predict which items will be interesting to the user. Given an initial set of preferences that is either explicitly provided by the users or is implicitly inferred by the system from user actions, the prediction model tries to estimate a function F for the user-item pairs for which preferences are not known yet.

$$F : User \times Item \rightarrow Utility$$

Here we call utility the user preference for the item. In an application where users rate items, it is possible to use the ratings as utility measurement [8]. For example, we can assume that a movie that the user rates with five stars has a higher utility for the user than a movie that the user rates with four stars. In another application where only user actions are collected, the

utility can measure how suitable an item is for the user to perform a target action on it [38]. Analogously to the previous example, one can assume that a movie that the user has watched has a higher utility for the user than a movie that the user did not watch. For a given user, RSs provide suggestions for items with the highest estimated utility.

Traditionally, the input to action prediction models is a set of past user-item interactions which can be ordered and timestamped. The interactions can be of a single [38, 76, 84] or of multiple types [31, 61, 91], they can be grouped into one meta-action (e.g., a sequence of clicks on an item can be grouped into one observation) [55], and they are usually associated with one of the recommendable items through numerical values r_{ui}. These values may represent the number of times user u purchased item i, the percentage of video i watched by user u, or even a rating given by user u to item i. Finally, 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) [82].

The output of action prediction models is usually associated with a target action and is either a predicted numerical value $\hat{r}_{ui} \in \mathbb{R}$ (e.g., expected video watch time [15], utility score [84]) or a predicted indicator value $\hat{p}_{ui} \in \{0, 1\}$. The *indicator* value p_{ui} expresses whether u will perform a target action on i or not [38].

Depending on the input and output, we identify four major groups of action prediction models: *basic collaborative filtering*, *sequence-aware*, *hybrid*, and *multi action type prediction models*.

2.2.1 Basic Collaborative Filtering Models

Initial works on predictions employing implicit feedback were trying to reuse prediction models designed for explicit feedback. The largest number of these models is based on Collaborative Filtering (CF) techniques [52] which predict unobserved user-item interactions, such as ratings or target actions, by exploiting user-to-user or item-to-item similarities. CF techniques can be grouped into two general categories of memory-based and model-based methods [71]. The prediction models presented in this section are often used as the basis for creating novel recommendation approaches.

Memory-based Methods

In memory-based CF, the observations stored in the system are directly used to predict unknown observations. For example, from the knowledge of the past purchases of certain items by certain users, the model can predict whether a target user will purchase a target

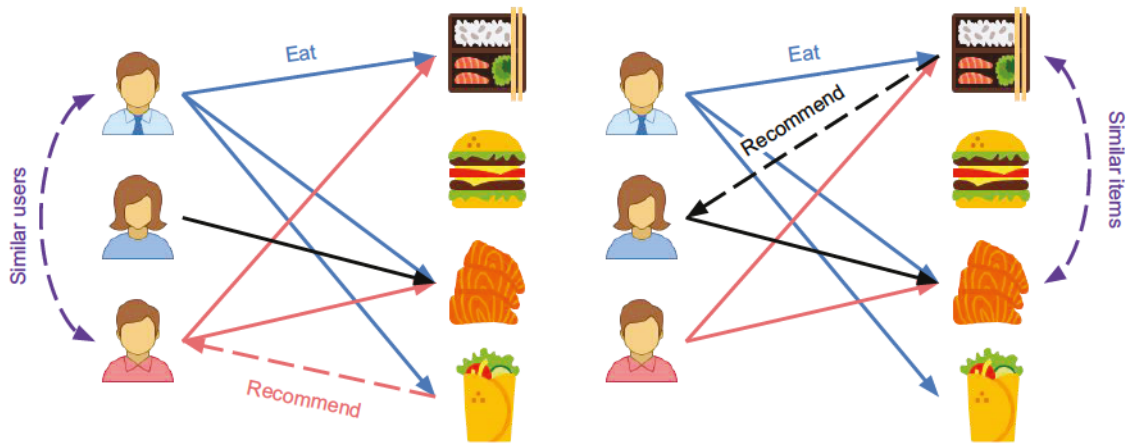

Figure 2.1: Examples of user-based (left) and item-based (right) CF approaches

item. The prediction can be done in two ways known as user-based [49] or item-based [17] approaches. User-based approaches evaluate the interest of a target user for an item using the user-item interactions for this item generated by other users, called neighbors, that have similar interaction patterns. The neighbors of the target user are typically the users whose interactions are mostly correlated to the target user's interactions. Item-based approaches, on the other hand, predict the interaction of a user with an item based on the interactions of the user with similar items. In such approaches, two items are similar if several users of the system have interacted with these items in a similar fashion (Fig 2.1).

In the case when the number of users exceeds the number of available items, item-based approaches are preferred since they provide more accurate recommendations, while being more computationally efficient and requiring less frequent updates [71]. On the other hand, user-based methods usually provide more original recommendations, which may lead users to a more satisfying experience [18].

Central to most item-based approaches is a similarity measure between items, where sim_{ij} denotes the similarity of items i and j. Frequently, it is based on either the Pearson correlation coefficient or cosine similarity [4]. The goal of the item-based CF (IBCF) model is to predict unobserved values r_{ui}. Using the similarity measure, the model identifies the k most similar items to i with which u interacted. This set of k neighbors is denoted by $S^k(i,u)$. In the simplest case, the predicted value of r_{ui} is calculated as a weighted average of the observed interaction values for neighboring items [71]:

$$\hat{r}_{ui} = \frac{\sum_{j \in S^k(i,u)} sim_{ij} r_{uj}}{\sum_{j \in S^k(i,u)} sim_{ij}}$$

Memory-based methods are intuitive and relatively simple to implement. In their simplest form, only one parameter (the number of neighbors used in the prediction) requires tuning. Moreover, unlike most of the model-based methods, they require no costly training phases and they are little affected by the addition of users, items, and interactions. At the same time, finding neighbor users or items can be computationally expensive, which in turn can affect the speed of the prediction. Therefore, the set of neighbors for each user or item is either pre-computed offline or is found by using approximate nearest neighbors methods, such as Annoy[1]. Once item neighbors and their similarities have been found, an item-based model can readily make predictions to new users, without having to re-train the model.

While memory-based methods have gained popularity due to these advantages, they are also known to suffer from the problem of limited coverage, which causes some items to be never recommended [71]. Also, traditional methods of this category are known to be more sensitive to the data sparsity and the cold-start problem, where the system receives only a few interactions, or no interactions at all, for new users and items. To overcome these problems more advance memory-based techniques have to be applied [71].

Model-based Methods

In contrast to memory-based methods, which use the stored interactions directly in the prediction, model-based approaches use these interactions to learn a set of model parameters (e.g., latent features of the users and the items). The parameters, which capture salient characteristics of users and items, are later used for making predictions [52]. The crucial characteristic of implicit feedback data is that all the unobserved user-item interactions are a mixture of actually negative and missing positive values. Therefore, to train a prediction model, the unobserved user-item interactions should be preprocessed. Two extreme preprocessing strategies are "All Missing As Unknown" (AMAU) and "All Missing As Negative" (AMAN) [76].

In AMAU, the missing data points are treated as unknowns. Therefore, a prediction algorithm only operates on the positive examples and ignores all the missing data. Since the whole dataset only consists of positive examples, typical machine learning models are unable to learn how to distinguish between the positive and negative examples. In this setting, the problem is similar to the *one-class classification* problem [54], which can be attacked with particular models, such as one-class SVMs [76]. These models learn to classify new data as similar or different to the training set. For instance, for the target user, the classifier predicts whether the target item is similar to other items the user positively interacted with in the past.

[1] https://github.com/spotify/annoy

The main disadvantage of such models is that they aim at learning one single concept, i.e., one has to create a classifier for every user.

In the AMAN approach, all the missing data points are treated as negatives. Since there are only few positive observations in the dataset, the approach biases a prediction model towards the negative feedback (the class imbalance problem [3]). Moreover, during the training step, all the user-item pairs for which the model should provide predictions are presented to the learning algorithm as negative examples. Therefore, a model with enough expressiveness (that can fit the training data exactly) will predict only negative scores [84]. To overcome these drawbacks and to balance the extent of treating missing values as negative examples Pan et al. [76] proposed two techniques. These techniques allow us to tune the trade-off in the interpretation of negative examples and result in better performing CF algorithms overall.

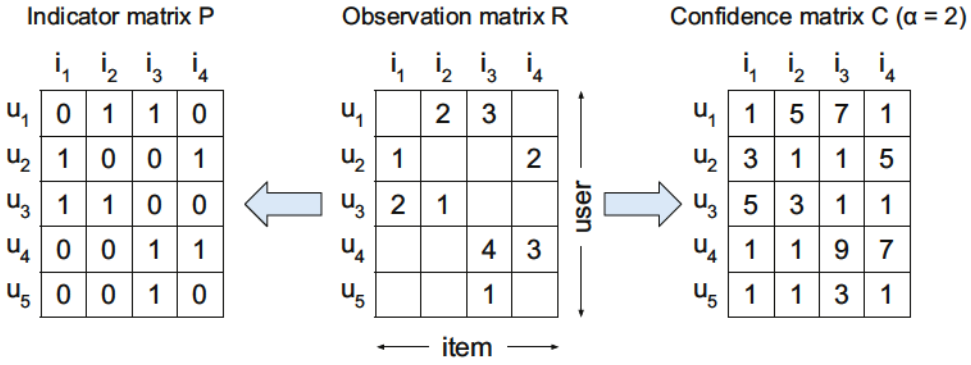

Figure 2.2: Examples of indicator, observation, and confidence matrices

The first technique is based on weighted low rank approximation [95]. Weighted-Regularized Matrix Factorization (WR-MF) [38, 76] is the most commonly used implementation of this technique. It is also a state-of-the-art implicit feedback CF model. WR-MF predicts whether a user will perform a target action on an item. The input data for the model is a user-item non-negative matrix R, whose entries r_{ui} count the number of observed actions of u on i. To predict whether a user will act on an item the *indicator* and *confidence* values are introduced (Fig. 2.2). The indicator value p_{ui} shows whether the user u has performed the target action on the item i ($p_{ui} = 1$) or not ($p_{ui} = 0$). The confidence value c_{ui} reflects how confident we are about each p_{ui} entry.

During the training phase $p_{ui} = 1$ if $r_{ui} > 0$, otherwise $p_{ui} = 0$. If $p_{ui} = 1$, we are confident that it is true, as it is based on real observations of the user u acting on the item i ($r_{ui} > 0$). Conversely, when $r_{ui} = 0$ the model should not be confident that u will not act on i in the future. WR-MF computes a confidence value c_{ui} on the base of r_{ui} as well. The state-of-the-art

choice for confidence estimation is:

$$c_{ui} = 1 + \alpha r_{ui}$$

where $\alpha \geq 1$ is a hyper-parameter that must be optimized. Increasing α places (proportionally) more weight on the non-zero observations while decreasing α places more weight on non-observed items. The target action prediction is then computed using MF: each user u and item i is associated with an f-dimensional factors vector $x_u \in \mathbb{R}^f$ and $y_i \in \mathbb{R}^f$ respectively. The predicted value of the indicator function is computed by the dot product of these two vectors:

$$\hat{p}_{ui} = x_u^T y_i$$

The vectors' parameters are learned by minimizing the following cost function:

$$\min_{x*,y*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

The constant λ is used for tuning the amount of regularization and avoid model over-fitting [79]. Cost function minimization is achieved by alternating least squares (ALS) optimization [38, 76]. The values of α and λ are data-dependent and determined by cross-validation.

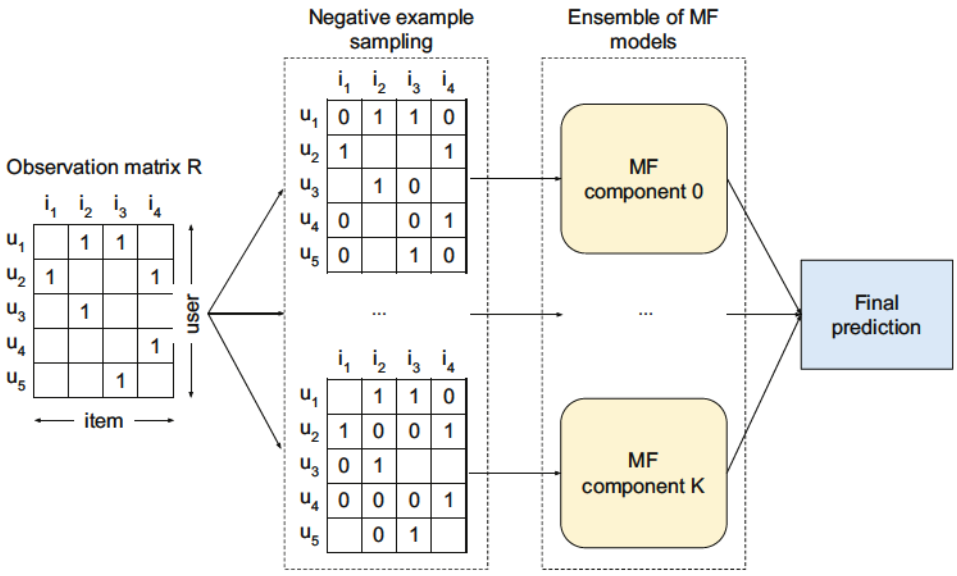

Figure 2.3: Graphical representation of the bagging technique for the target action prediction task

The second technique is based on negative examples sampling (Fig. 2.3). The implementation of this technique, proposed by Pan et al. [76] is based on the ensemble method called bagging [9]. Each component of the ensemble is a MF model which is trained using all the positive observations and randomly sampled missing observation considered as negative examples. The ensemble predicts the indicator value $\hat{p}_{ui}$ by averaging $\hat{p}^k_{ui}$ values predicted by each component model k.

$$\hat{p}_{ui} = \frac{1}{|K|} \sum_{k \in K} \hat{p}^k_{ui}$$

Model-based approaches are the state-of-the-art in CF. Over the years, a wide variety of latent factor models have been proposed based on the choices of the objective function and optimization constraints [52]. These models have different trade-offs in terms of prediction accuracy, stability, complexity, and interpretability. It has been shown that model-based methods have superior performance compared to memory-based approaches [53]. However, similarly to memory-based approaches, the basic model-based methods suffer from the data sparsity and the cold-start problem [106].

Ranking Optimization

The task of item recommendation is to create a user-specific ranking for a set of items. Even though all the previously presented methods are designed for the item utility prediction task of personalized ranking, none of them is directly optimized for ranking. For this reason, Rendle et al. [84] proposed the generic optimization criterion BPR-Opt which can be applied to the existing models, such as MF or IBCF, to optimize them directly for ranking. The criterion directly optimizes the AUC ranking metric [40].

The authors of BPR-Opt hypothesize that a user prefers items which she interacted with to items she has never interacted before. For example, if user u has viewed item i_2 but not item i_1, then u prefers item i_2 over i_1: $i_2 >_u i_1$. The task of personalized ranking is to provide user u with a ranked list of items, such that for all pairs of items $(i, j) \in I^2$ the personalized pairwise order $i >_u j$ has to be satisfied if i is preferred more than j.

BPR-Opt and its modifications are widely used in RSs [34, 84, 102]. In particular, there are several multi action prediction models which employ the BPR-Opt criterion [55, 61]. We present these models in Section 2.2.4. In addition to BPR-Opt, there are also other pairwise optimization criteria, for example, TOP1 [34]. Models based on these criteria belong to the class of pairwise learn to rank methods [59].

2.2.2 Sequence-aware Models

In many application domains, especially those employing implicit feedback, multiple user-item interactions can be recorded over time. For instance, a user can visit the same web-page or purchase the same product multiple times. This information can be used to build richer user models and to discover additional behavioral patterns that can be leveraged in the recommendation process [82]. In particular, it has been recently showed that the use of temporal patterns and information about the ordering of actions can improve the accuracy of action prediction models [35, 44].

The input to sequence-aware prediction models is the ordered and often time-stamped sequence of past user actions [82]. The content of the input sequence may vary depending on the task. For example, the playlist continuation task may require a sequence of all the tracks that the user listened to during the current session. At the same time, a log of actions performed by a user on an item may suffice to predict whether the user will perform a target action on that specific item.

Generally, sequences of user actions can be be represented in two forms: as user-item interaction histories or as user interaction histories. A *user-item interaction history* h_{ui} is the chronologically ordered sequence of all the actions performed by u on i. Similarly, a *user interaction history* h_u is the chronologically ordered sequence of all the actions performed by u over time. When user-item histories are used, a user's interactions with different items are considered independently from each other. In this case, if the user will first interact with item A, then with item B, and then again with item A, the model will not be able to understand that the user's second interaction with A might be influenced by the interaction with B. Despite the obvious limitation of user-item histories, they can be used with simple and computationally inexpensive action prediction models.

Moreover, the interaction histories can be either long- or short-term. Long-term histories usually contain actions performed by users over the whole period of life on some service, while short-term histories can contain only a few actions observed within the latest interaction session.

Depending on the available amount of histories, their representation, and the problem settings, different types of sequence-aware models are used for the action prediction task [82]. Below we briefly present three classes of sequence-aware techniques, namely *frequent pattern mining*, *sequence modelling* and *distributed representations*. It is worth noting that in this section, we mainly focus on techniques developed for single action type interaction histories. Details on the application of sequence-aware techniques to multi action type interaction histories are presented in Section 2.2.4.

Frequent Pattern Mining

Frequent Pattern Mining (FPM) techniques were initially developed to discover user consumption patterns within large transaction databases [82]. These techniques focus on identifying items that frequently co-occur in the same transaction [1, 2]. The obtained co-occurrences (patterns) are used to find items that best complement a given item or a set of items. Later, FPM was adapted to other domains, such as web personalization, music and e-commerce [7, 44, 68].

In most cases, the patterns are mined in an offline process and usually translated into a set of association rules or another compact form of knowledge representation [82]. Each resulting representation is associated with values expressing its strength (e.g., confidence and support) [1]. These values are used during the prediction phase to identify the best co-occurring items.

FPM techniques are well-explored and also easy to implement and interpret [82]. They can be applied to user action observations of both single and multiple types. For example, in the e-commerce domain, one can expect that since *add-to-cart* and *purchase* actions are usually observed nearby, there is a high probability that a user will purchase an item that she put in her shopping cart. Among the drawbacks of FPM techniques, one can highlight the limited scalability of some approaches and the general problem of finding suitable threshold values controlling the number of generated rules.

Sequence Modelling

Sequence modelling techniques aim at learning models which based on the past observations predict future ones, which in our case are user actions. The resulting models, some of which have recently established new state-of-the-art algorithms for RSs, can take into account user actions of single or multiple types. The most commonly used sequence modelling techniques are Markov Models, Recurrent Neural Networks, and Reinforcement Learning.

Markov Models consider the sequential interaction of users as a stochastic process over discrete random variables (states) related to predefined user behavior. The Markov property limits the dependencies of the process to a finite history [82]. The central assumption of Markov-Model-based approaches in the context of sequence-aware action prediction models is that the next user actions depend only on a limited number of the most recent preceding actions. Shani et al. [89] were among the first who applied Markov Chains (MCs) in RSs and showed the superiority of sequential models over non-sequential ones. In the music domain, McFee et al. [64] proposed a music playlist generation algorithm based on MCs that given a seed track predicts the next track to play. The main issue with applying MCs in

sequence-aware models is that the state space quickly becomes unmanageable when trying to include all possible sequences of user actions [35].

Recurrent Neural Networks (RNNs) are a class of artificial neural networks that can use their hidden state (memory) to process sequences of inputs. This makes RNN-based models applicable to sequence-related tasks such as next action or item prediction. At each time step, the hidden state of the RNN is computed from the current time step input and the previous time step hidden state. Thus, by memorizing the influence of all the past inputs, the hidden state makes RNNs well-suited for modelling the complex dynamics of user action sequences. Moreover, it helps to overcome the fundamental limitation of Markov-Model-based approaches. Variants of RNNs such as LSTM [16] and GRU [100], through their sophisticated hidden dynamics, can model much longer and complex temporal dependencies compared to other approaches like Hidden Markov Models [12].

RNN-based techniques have been increasingly explored in the past few years. For example, Zhang et al. [105] used RNNs for predicting clicks in sponsored search. The proposed model is trained to predict the next click of the users given their previous clicks. To model user activity in a session-based scenario in the e-commerce and media domains, another RNN-based action prediction model has been proposed by Hidasi et al. [35]. Similarly to the previous example, the model is trained to predict the next item in a sequence given the current one. Different loss functions have been introduced in the paper, which at the end led to better prediction performance. Later on, Hidasi et al. [36] included item features into the sequence model which further improved the prediction accuracy. To model user preferences across sessions, Quadrana et al. [83] proposed a model employing hierarchical RNNs. The first level of the hierarchy, which represents the state of the user across sessions, is used to initialize and propagate information to the second level of the hierarchy, which is used to generate recommendations within a session. Transferring the information from prior sessions ultimately led to better recommendations. Finally, Song et al. [94] proposed a context adaptation approach to build a session-aware recommender that can handle long-term (e.g., seasonal) and short-term changes in user preferences in the news domain.

A potential issue with RNN approaches is that a neural network needs to be able to compress all the necessary information of an input sequence into a fixed-length hidden state. This may limit the capability of the neural network to cope with long sequences, even when LSTM or GRU cells are used [5]. To extend the capabilities of RNN, different extensions, such as Neural Turing Machines [26] and Attentional Interfaces [5], have been proposed [74]. These extensions, while effective, increase the computational costs of the RNN-based models. Thus the computational demands for testing and optimizing multiple hyper-parameters of deep learning approaches still prevent their widespread use and exploration.

Reinforcement Learning (RL) techniques learn to model sequences by interacting with the environment, which is often formulated as a Markov decision process (MDP). In a recommendation scenario, the interaction is a recommendation of an item to the user (the action) for which the system then receives a feedback (the reward) [82]. For instance, in the music domain, the RL-based system first recommends a track to a user and then monitors whether the user will listen to it or skip it. A positive reward is assigned to the system if the user listens to the track and zero otherwise. The goal of the system is to maximize the cumulative reward computed over a number of interactions [99]. An example of a sequence-aware action prediction model using RL is an MDP-based model introduced by Shani et al. [89] for an online bookstore. The model predicts the user's next book selection based on her previous selections. By using RL, the model not only tailors recommendations to the recent user activity but also to the expected reward (income) for the shop.

Recent works in the field of machine learning and artificial intelligence have shown that by applying RL, one can create highly efficient generative models that can imitate the behavior of sophisticated agents such as humans [67, 92]. Nevertheless, the creation of RL-based models is still computationally expensive and requires an accurate description of the environment.

Distributed Representations

Distributed representations are dense, lower-dimensional vector representations of the objects (e.g., users, items, or action types) where the "meaning" of an object is distributed across multiple vector components. Methods that learn distributed representations from sequences of user actions associate each object with a real-valued embedding vector, which represents the object's projection into a lower-dimensional space in which the sequential relationships between the objects are preserved. For example, some item representations translate pairwise transition probabilities into distances in a Euclidean space [11]. Thus, in a sequence-aware recommendation problem, one can then recommend the next items by searching the nearest neighbors to the last items explored by the user [27].

Distributed item representations have been successfully applied in many domains. For example, Feng et al. [21] used them for building personalized POI recommendation based on the last-N POIs visited by the user. To deliver personalized product ads, Grbovic et al. [27] proposed another approach based on the "distributional hypothesis", which states that semantically equivalent words frequently occur in the same contexts. The proposed Prod2Vec model learns distributed item representations from sequences of emails containing purchase receipts. Specifically, Prod2Vec uses the skip-gram model [66] to project items that tend to have similar neighboring items in the receipts to close to each other. Given the sequence of

the last few observations for a user, the most similar items in the lower-dimensional space represent the recommendation candidates. An enhanced version of the skip-gram model that conditions the item embeddings also on their metadata has been proposed by Vasile et al. [103].

While embedding methods can exploit the sequential relations between objects to learn their representations, these methods do not make direct use of the sequence of recent user interactions to generate the recommendations [82]. Instead, they rely on approximate methods, like nearest neighbors, to predict the next interactions of the user. This in turn can lead to limited effectiveness in domains with strict ordering constraints for which sequence modelling methods can be preferable [82].

2.2.3 Hybrid Models

Each RS technique has its own strengths and weaknesses. In general, in a target application domain one would like to make use of all the knowledge available in different data sources and also use the algorithmic power of various techniques to improve recommendation performance. To fulfill these goals, researchers designed hybrid recommender systems (HS) which combine several techniques to minimize their inherent drawbacks and, most importantly, to maximize their advantages. Burke distinguishes seven hybridization techniques [10].

Weighted: The scores of different recommendation components are combined into a single weighted score.

Switching: The system selects and applies a single recommender from among its constituents based on the current recommendation situation.

Mixed: Recommendations generated by different recommendation components are presented side-by-side in a combined list.

Cascade: Recommenders are given strict priority, with the lower priority ones breaking ties in the scoring of the higher ones.

Feature Combination: Features derived from different recommenders are combined and injected into a single recommendation algorithm.

Feature Augmentation: A feature or set of features computed by one recommendation technique are passed as input to the next technique.

Meta-level: The model learned by one recommender is used as input to another.

Though HSs usually combine the power of different types of models, e.g., Collaborative and Content-based Filtering [88], such systems can also incorporate models of the same type. For example, several homogeneous component recommenders can be combined into an ensemble model using boosting techniques [60, 104]. The component recommenders are based on a fixed CF algorithm using a re-weighting strategy, which assigns a dynamic weight distribution on the observed user-item interactions. Each subsequent component improves the previous ones by adjusting weights of misclassified interactions.

HSs are often designed in such a way that a model based on one technique complements or improves the performance of a model based on another technique. A similar hybridization approach has been used in some of the models proposed in this thesis. As an example, let us consider the HS proposed by Rendle et al. (FPMC) [85] which employs tensor factorization to learn personalized Markov chains. FPMC uses the BPR optimization criterion [84] to predict the unobserved entries in the sparse tensor, i.e., to predict personalized transitions between pairs of items. At recommendation time, the items are ranked according to their likelihood to be the next item given the last item the user has interacted with.

Another similar HS was introduced by Melville et al. [65]. The Content-Boosted Collaborative Filtering (CBCF) hybrid approach incorporates Content-based (CB) and CF models to provide a user with movie recommendations. The CB component exploits user and item features to densify the original user-item ratings matrix by predicting the unknown user-item ratings. Then, the densified matrix is used by the CF component to generate recommendations.

Due to the capability of hybrid approaches to overcome the shortcomings of individual methods they are often used to build RSs [10]. The typical challenges when designing HSs include the problem of determining the optimal combination of the considered components and finding the best way of merging the predictions of the different components into the final prediction.

2.2.4 Multi Action Type Prediction Models

Though the single action type models provide reliable results, the availability of additional information (e.g., explicit feedback) and multiple action types makes it possible to improve action prediction accuracy [51, 78]. The collection of multiple action types elicits more information about a user behavior and can be jointly used to build more comprehensive user models [58, 77, 81, 101]. The resulting models can better describe users' preferences and can generate better recommendations.

To illustrate various aspects of the design of multi action prediction models, we introduce several of them below. We start with simple heuristic-based models and then present

more complex sequence-aware prediction models incorporating additional information such as context or item features. Some of the presented models are used as baselines in our experiments. When we describe these models, we cite the thesis chapters where they have been compared to our proposed ones.

Lin et al. [57] studied a situation where a user sees an item but does not click on it. The authors conjectured that such user behavior might be considered as a negative preference signal for the item. At the same time, if a user clicks on an item then the target action is observed. Using these two types of actions (i.e., clicks and impressions) they extended the WR-MF model [38, 76] by introducing two confidence functions separately calculating confidence values for positive and negative observations. Empirical evaluations showed that the proposed approach outperforms models employing only single-type implicit feedback in terms of prediction accuracy.

Although the model above incorporates two types of user actions, it is hardly extensible to multiple action types, especially if action types have the same polarization. For example, in the purchase prediction task, clicks and bookmarks can be both considered as positive signals. Hence, one cannot train the confidence function responsible for the negative feedback.

Loni et al. [61] conjectured that different types of user actions performed on an item, e.g., a click or a like, reflect different "levels" of commitment or preference. Based on this assumption the authors proposed a pair-wise method called Multi-Feedback Bayesian Personalized Ranking (MF-BPR) which exploits multiple types of user actions. In MF-BPR, each action type is associated with a preference level, while the optimal relative ordering of the levels is selected manually. The levels are used to sample item pairs such that the first item in the pair is preferred to the second item. The resulting pairs are then used to train the original BPR model [84]. Thus, the proposed approach takes advantage of the available information consistently with the intuition that some user feedback signals are more reliable or meaningful than others. Experiments, which were carried out on three datasets containing multiple types of feedback, demonstrated that MF-BPR can outperform BPR in terms of prediction accuracy. Similar models have also been proposed by Lerche et al. [55] and Pan et al. [77]. We used MF-BPR as baseline model for the evaluation of the AMMF model introduced in Chapter 5.

It is not always clear how to interpret user actions. For example, in the music domain, a skipped track may, but need not, indicate a negative rating. Furthermore, the interpretation may be user-dependent. In their paper, Kordumova et al. [50] proposed a method to learn, from explicit ratings, how to translate user actions on items to ratings on these items. They do this by associating user actions to rated items through manually engineered item features, such as "the number of times the track has been played" or "the time of the day the track was

played". Naive Bayesian classification is then used to rate new items with which the user has interacted. These implicitly rated items provide an additional source of information for recommending new items. The evaluation, which was conducted on data from a web-based music player, showed that the method enables a form of implicit learning that adapts to each user individually.

Shi et al. [91] studied the personalized ranking problem by integrating one type of explicit feedback and multiple types of implicit feedbacks. The proposed personalized ranking framework MFPR takes into account relationships between explicit and implicit feedbacks as well as between multiple types of implicit feedback. Extensive experiments showed that the integration of multiple feedback types significantly improves the recommendation model performance. To use this model, explicit user feedback has to be provided.

Finally, Smirnova et al. [93] and De Boom et al. [16] proposed RNN-based multi action type prediction models leveraging sequences of user-item interactions. The authors consider action types and timestamps as the context in which user-item interactions have been observed. This contextual information is used as a part of the input to an RNN model. The action types are modelled with one-hot encoded vectors. The Concat-Mult-GRU-RNN model proposed by Smirnova et al. [93] was used as baseline model for the offline evaluation of our RNN-based model called LSTP, which we introduce in Chapter 5.

Empirical evaluations of the presented models showed that using actions of multiple types is beneficial. Moreover, in terms of prediction accuracy, such models can be superior to those employing single-type implicit feedback. However, the domain specificity, which imposes limitations on the range of the relevant user actions, is one of the main limitations of the cited works. Usually, the selection of the predictive user actions is done manually by using different heuristics that may not take into consideration subtle relations between actions of different types. The absence of such information may decrease the quality of the prediction model.

It is also worth to mention that all the presented models aim to capture mainly long-term user preference profiles. However, it has been recently shown that a combination of short- and long-term interests can be central to the success of a recommender [42, 83].

2.3 Conclusions

Over the past three decades, a variety of action prediction models have been developed for implicit feedback RSs. Generally, in the context of user profiling, researchers rely on one type of specific user interactions like item view events or purchases. However, the interactions between a user and an item are rarely limited to a single type of actions and, usually, much

richer types of information are available. For example, multiple additional actions related to certain items (e.g., add-to-wishlist, add-to-cart in the e-commerce domain) can provide a valuable piece of information.

Although several models incorporating actions of multiple types have been recently proposed in the research literature, most of them are based on domain-dependent heuristics which may overlook subtle relations between actions of different types. Moreover, these models capture only long-term user preference profiles and do not take into account temporal delays between actions or the order of actions. Therefore, the original contribution of this thesis is a set of domain-independent multi action prediction models for RSs that can automatically elicit predictive action types for a pre-defined target action type, determine correlations between action types, and benefit from the sequence- and time-awareness.

Chapter 3

General Action Prediction Model

State-of-the-art implicit feedback models presented in Chapter 2 predict whether the user will perform a target action on an item. These models are usually trained by using the observations of user actions of one single type. For instance, they predict that a user will watch a video using a dataset of observed video watch actions.

In this chapter, we present a general prediction model (MMF - Multiple action types Matrix Factorization) which is designed to predict a target user action by leveraging information about actions of multiple types. Our model, which is based on Weighted-Regularized Matrix Factorization (WR-MF) [38, 76], takes advantage of the existence of a range of action types to improve the confidence in certain observations for a given target action. For example, if a user opened the description of a movie, read it, watched a movie trailer but did not watch the movie, then the method increases the confidence that the user will not watch this movie. The empirical evaluation of MMF showed that using multiple actions is beneficial and can outperform WR-MF that uses only the target action data. The model is used as the basis for the more advanced approaches presented in the following chapters.

It is worth noting that the analysis of MMF, which is presented in this chapter, was made on proprietary data to which we no longer have access to. For this reason, in this chapter, we present the results of the empirical evaluation published earlier in [32]. That is why the protocol and metrics presented in Section 3.3 slightly differ from those used in the rest of the thesis. Nevertheless, we do not consider this as a significant limitation, since in the following chapters we evaluate MMF on other datasets and compare it with other models.

3.1 General Model and Examples

Let us consider a dataset of $d+1$ types of user actions. The variable r_{ui}^{j} counts or measures the observations of an action of type j performed by a user u on an item i. We assume

that there is a target action $j = 0$ (predicted), and non-target actions are instead those for $j \in \{1, \ldots, d\}$ (predictive).

Similarly to WR-MF, the indicator function of the target action p^0_{ui} and the confidence c^0_{ui} of the target action are introduced. However, they are now functions of all the observations for all the action types:

$$p^0_{ui} = p(r^0_{ui}, r^1_{ui}, \ldots, r^d_{ui})$$
$$c^0_{ui} = c(r^0_{ui}, r^1_{ui}, \ldots, r^d_{ui})$$

The target action prediction model is build by using matrix factorization (MF). The predicted value of the indicator function is computed as s dot product of f-dimensional user and item factors vector $x_u \in \mathbb{R}^f$ and $y_i \in \mathbb{R}^f$:

$$\hat{p}^0_{ui} = x_u^T y_i$$

The vectors' parameters are learned by minimizing the following objective function:

$$\min_{x*, y*} \sum_{u,i} c^0_{ui} (p^0_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

The difference, with respect to WR-MF, is due to the usage of different confidence and indicator values, i.e., c^0_{ui} and p^0_{ui}.

Below we illustrate two applications of MMF related to P7S1's video streaming service 7TV[1] that provides access to videos and TV shows. We consider three types of actions: a user opened a video details page, a user stopped to watch a video and a user marked a TV show as "watch later".

3.1.1 Predicting Video Views with Open Details Actions

In the first example MMF predicts whether a user will watch a video by also leveraging the observations of two additional action types: the user opened the video details page and the user stopped to watch a video.

Let $v_{ui} \in [0, 1]$ denote the percentage of video i viewed by user u. The service obtains this data when a user stops to watch a video. A positive example of the user target action (a user started to watch a video) is observed when $v_{ui} > 0$. So, in case only this target action type is

[1] https://www.7tv.de/

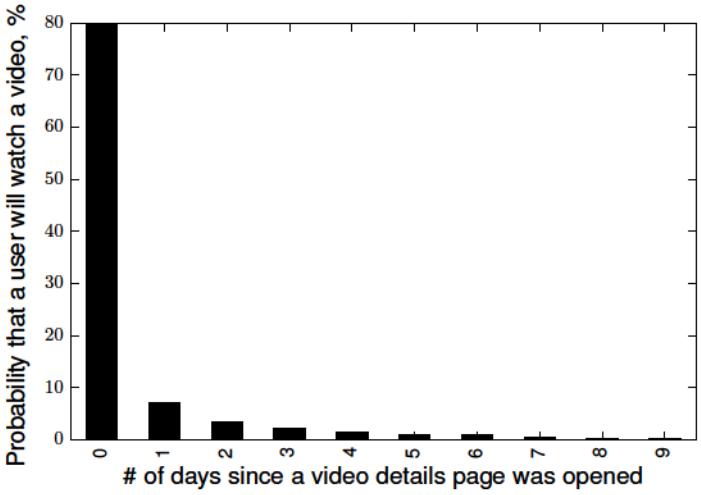

Figure 3.1: The distribution of probability that a user will start to watch a video after Δ days since the video details page was opened

used, the indicator function p can be defined as:

$$p_{ui} = \begin{cases} 1 & v_{ui} > 0 \\ 0 & v_{ui} = 0 \end{cases}$$

In general, as v_{ui} grows, one have stronger confidence that $p_{ui} = 1$. At the same time, for items where observations are missed the confidence that $p_{ui} = 0$ should be much smaller. Like in WR-MF, a feasible choice for c_{ui} would be:

$$c_{ui} = 1 + \alpha v_{ui}$$

The parameter $\alpha \geq 0$ is a hyper-parameter that must be optimized.

Analyzing the data provided by P7S1 we noted that if a user opened a video details page but did not watch the video, then there is a high probability that the user will not watch the video in the future. The probability is proportional to the time passed since the video details page was opened (Fig. 3.1). This observation can be used to increase the confidence in the negative examples ($p_{ui} = 0$) of the target action. The more days are passed since the video details page was opened (the video still has not been watched), the higher must be the confidence that p_{ui} will stay equal to 0.

Let us denote with o_{ui} the variable indicating that a user has visited ($o_{ui} = 1$) or not ($o_{ui} = 0$) a video details page. Moreover, let Δ_{ui} be the number of days since the visit action, $o_{ui} = 1$, was observed, and $b^{-\Delta_{ui}}$ is the function that approximates the probability distribution displayed

in Fig. 3.1. Then the confidence value is assumed to be as follow:

$$c_{ui} = \begin{cases} 1 + \alpha v_{ui} & v_{ui} > 0 \\ 1 + \beta\left(1 - b^{-\Delta_{ui}}\right) & v_{ui} = 0 \; and \; o_{ui} = 1 \\ 1 & v_{ui} = 0 \; and \; o_{ui} = 0 \end{cases}$$

Here α, β and b are data-dependent hyper-parameters that should be tuned. The values of α and β should be greater than or equal to 0, while the parameter b should be greater than 1.

3.1.2 Predicting TV show Views with Watch Later Actions

In the second example MMF predicts that a user will watch a TV show. Even in this scenario observations of two action types are used: a user watches a TV show, and a user marks a TV show as "watch later". Let us denote with $s_{ui} \in [0, 1]$ the percentage of a TV show i viewed by a user u. The positive examples of the user target action (a user started to watch a TV show) are observed when $s_{ui} > 0$.

In contrast to the previous example, the "watch later" action type can be here used to reinforce the confidence in positive examples of the target action. In fact, in the dataset in 86% of cases a user watched a TV show after she marked it as "watch later". Let $l_{ui} \in \{0, 1\}$ indicate the absence/presence of item i in the set of TV shows that u would like to watch later. According to the data analysis, if a user adds an item to the "watch later" set, then there is a high probability that a user will watch such an item. Hence, the indicator function p can be defined as:

$$p_{ui} = \begin{cases} 1 & s_{ui} > 0 \; or \; l_{ui} = 1 \\ 0 & s_{ui} = 0 \; and \; l_{ui} = 0 \end{cases}$$

At the same time, the confidence that a user will watch a TV show should be increased when a user marks it as "watch later". So, the confidence value can be defined as:

$$c_{ui} = \begin{cases} 1 + \alpha s_{ui} + \gamma l_{ui} & s_{ui} > 0 \; or \; l_{ui} > 0 \\ 1 & s_{ui} = 0 \; and \; l_{ui} = 0 \end{cases}$$

As in the previous example, α and γ are hyper-parameters that must be determined by cross-validation. Both of them should be greater than or equal to 0.

Table 3.1: Statistics of the used datasets

Collection name	Size
Devices	127609
Videos	23006
TV shows	347
Video details opened action	978625
Video view stopped action	800872
Watch later action	1096

Table 3.2: "Video view stopped action" record

User	Video	Watched percent	Timestamp
4E6E8D...	380934	0.84	1438387203

3.2 7TV Dataset

The dataset provided by mass media company P7S1 has been used for the empirical evaluation of the proposed model. This dataset contains information about the interaction of users with the online video streaming service 7TV for one month. A user interacts with the service through a set of devices. For this reason, we consider a device as a user, i.e., users are anonymized. Table 3.1 shows some statistics about the users (devices), items (videos and TV shows), and observations of different action types.

A "Video details opened action" record indicates that a user opened a video details page. A "Video view stopped action" record contains the maximal percentage of a video viewed by a user. A "Watch later action" record refers to a TV show that a user marked to watch it later. All action records contain the time when the action was performed. For the "Video view stopped action" type the *timestamp* field stores the last time a user watched the video (Table 3.2).

"Videos", "TV shows" and "Devices" in Table 3.1 count only items that are found in the actions collections. The sparsity of the user-item indicator and confidence matrices are approximately 99.9%.

3.3 Evaluation

In our experimental evaluation, MMF has been compared with WR-MF in terms of Mean Percentage Ranking (MPR) [38, 46] and Recall at k (R@k) [40]. The original data has been split into the training and testing sets using a time-dependent splitting criterion. The first three weeks were used for the training, while the more recent actions were used for validation and

testing purposes. In particular, the first day of the holdout data was used for validation and the remaining days for testing. The hyper-parameters were optimized by benchmarking the trained models on the validation set. When hyper-parameters were identified, the validation set was merged with the training set, and the models were retrained on the merged data. We indicate with T the testing set of target actions, and the observation of an action in T is denoted with a^t_{ui}. In our datasets, users often perform a target action multiple times. Because it is more interesting for a user to be recommended with items that she has not interacted with recently, or that she is not aware of [38], we removed from the test set T all the actions a^t_{ui} belonging to the target type that were already presented in the training set. Additionally, users and items that were not present in the training set are removed from the testing set.

The models generate for each user a list of items sorted in descending order by their predicted value $\hat{p}_{ui}$. MPR evaluates a user's satisfaction with an ordered list of items [46].

$$MPR(T) = \frac{\sum_{u,i \in T} a^t_{ui} rank_{ui}}{\sum_{u,i \in T} a^t_{ui}}$$

We denote by $rank_{ui}$ the percentile ranking of item i within the ordered list of all the items predicted to receive the target action by user u. $rank_{ui} = 0\%$ indicates that i is predicted on top of the items that u will interact with, while $rank_{ui} = 100\%$ indicates that i is predicted at the bottom of the ranked items for u. Lower values of MPR are better, as they indicate that users actually interacted ($a^t_{ui} > 0$) with the clips that are ranked on top.

We also calculated Recall at k (R@k). Let us denote with M_u the set of top-k ranked recommended items for a user u and with T_u the set of items in the test data which the user u actually interacted with. If U^T is a set of test users, then Recall at k (R@k) can be defined as:

$$R@k = \frac{1}{|U^T|} \sum_{u \in U^T} \frac{|M_u \cap T_u|}{|T_u|}$$

In the evaluation experiments $k = 20$.

The comparison of the models has been done on the two applications presented above for the different number of latent factors $f \in \{10, 20, 30, 40\}$. In all the cases MMF outperforms WR-MF. That is why we present the results only for the optimal number of latent factors, $f = 20$, which was found by cross-validation on the training data.

In the first example the models predict that a user will watch a video, while in the second one the models predict that a user will watch a TV show. WR-MF is trained by using only the target action data, while MMF uses also the "open video details" and "watch later" actions. As can be seen in Table 3.3, MMF outperforms WR-MF in terms of MPR and R@20.

Table 3.3: Comparison of WR-MF and MMF ($f = 20$)

	Metric	WR-MF	MMF	Improvement
Video views prediction	MPR	12.9	12.6	2.3%
	R@20	17.1	17.3	1.1%
TV show views prediction	MPR	27.3	24.4	10.6%
	R@20	38.8	44.8	15.5%

One must observe that in the second example the "watch later" actions let us add positive examples of the indicator function, thereby increasing prediction recall. Conversely, in the first example by considering the "open video details" actions we can only tune the confidence of negative examples of the indicator function. That explains why the improvement brought by MMF is larger in the second case.

3.4 Conclusions

In this chapter, we have presented a general model (MMF) that predicts a target user action by leveraging information about actions of multiple types. The empirical evaluations of MMF showed that using actions of multiple types is beneficial and that MMF can outperform models that employ single-type implicit feedback. This conclusion was not obvious since not all available action types may be relevant for the prediction of a target action.

Similarly to Lin et al. [57], it has been shown that action types negatively correlated with a target action type can be successfully employed to predict a target action. At the same time, the ability of MMF to use actions of multiple types, even with the same polarization, makes it more general.

The incorporation of multiple action types into the prediction model requires the definition of specific indicator and confidence functions and the optimization of multiple hyper-parameters. Currently, these functions have been identified heuristically, after an exploratory analysis of the data. Such an approach may not take into account hidden relationships between actions of different types, which may decrease the quality of the prediction model. In Chapter 4 we show how by applying machine learning (ML) techniques one can automatically discover the relationships between actions of different types and compute the indicator and confidence values.

Chapter 4

Automated Multi Action Type Matrix Factorization

Multi action implicit feedback models leverage information about actions of various types to predict whether the user will perform a target action on an item. The growth of the number of action types makes it more challenging to explore manually which action types can predict a target action. However, the majority of the existing models use manually crafted domain-dependent heuristics to describe correlations between actions of target and non-target types [32, 50, 55, 57, 61, 82]. An example of such a model is Multi action type Matrix Factorization (MMF) presented in the previous chapter. MMF incorporates multiple action types through the definition of specific indicator and confidence functions. Initially, these functions have been identified heuristically, after an exploratory analysis of the data. In this chapter, we present an extension of MMF called Automated Multi action type Matrix Factorization (AMMF) which finds correlations between different action types automatically using machine learning (ML) techniques. The automation makes the process of finding correlations between action types easier and more precise, which in turn increases the prediction accuracy of the model. Moreover, it allows to take into account additional valuable information, such as temporal delays between actions or the order of actions.

4.1 Hybrid Prediction Model

In multi action environments, a user may perform multiple, possibly repeated, actions on an item. For instance, before an item is bought by a user, it might be searched or browsed by the same user several times. In this way, for each item, one can observe a sequence of different actions performed by a single user, which we refer to as a user-item interaction history h_{ui}.

Table 4.1: Crucial features of AMMF and its constituting prediction models

	$\mathcal{M}$	WR-MF	AMMF
Predicts actions of the target type	+	+	+
Employs actions of multiple types	+		+
Employs time and ordering of actions	+		+
Employs collaboration between multiple users and items		+	+

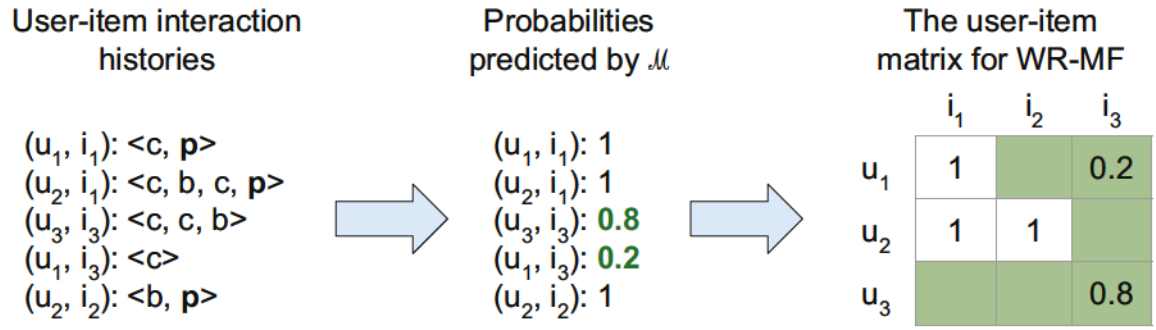

Figure 4.1: The prediction pipeline of the AMMF model

In the scope of MMF, one would like to create such indicator and confidence functions that $p_{ui} = p(h_{ui})$ and $c_{ui} = c(h_{ui})$ respectively. Similarly to Weighted-Regularized Matrix Factorization (WR-MF), if h_{ui} is empty, then one can assume with a low confidence that $p_{ui} = 0$. Analogously, if h_{ui} contains actions of a target type, then $p_{ui} = 1$ with the confidence proportional to the number of observations of the target actions in h_{ui}. For the non-empty histories without target actions, one should decide whether $p(h_{ui}) = 1$ and, if so, what is the confidence score for the resulting $p_{ui} = 1$. This decision can be done automatically by creating a probabilistic model $\mathcal{M}$ that predicts, for all the user-item histories that do not contain the target action a^t, the probability that the next unobserved user action will be of the target type. The estimated probabilities are then used to calculate the indicator and confidence values.

AMMF is a hybrid action prediction model [10] which consists of two components. The first component is the $\mathcal{M}$ model which predicts the probability that a user will perform a target action on an item given a user-item interaction history. This component can also take into account the ordering of the actions and the time delays between them. The second component, which is the WR-MF model, employs the first component and the "collaboration" between multiple users and items to produce the final action predictions. In particular, it uses the actual numbers of observations of the target actions and the estimated probabilities to calculate the indicator and confidence values. Table 4.1 illustrates features of AMMF and its components.

Fig. 4.1 illustrates on a tiny dataset, which consists of five user-item interaction histories, the prediction pipeline of the AMMF model. The dataset contains user actions of three types, namely clicks (c), bookmarks (b), and purchases (p). The purchase action type is the target one. For all the histories that are not ending with the target action, the $\mathcal{M}$ model predicts the probability to observe the target action. The estimated probabilities and the real observations of the target actions are then used to generate the user-item matrix. This matrix is used by WR-MF to produce the final action predictions for all user-item pairs that are not ending with the target action (highlighted in green).

Let us denote with $P(a^t|h_{ui})$ the probability that the next unobserved action in the user-item interaction history h_{ui} will be of a target type. Then, the indicator and confidence values can be identified as follows:

$$p_{ui} = p(h_{ui}) = \begin{cases} 1 & r_{ui} > 0 \\ 1 & r_{ui} = 0 \text{ and } h_{ui} \neq \varnothing \text{ and } P(a^t|h_{ui}) \geq \delta \\ 0 & \text{otherwise} \end{cases}$$

$$c_{ui} = c(h_{ui}) = \begin{cases} \propto r_{ui} & r_{ui} > 0 \\ \propto P(a^t|h_{ui}) & r_{ui} = 0 \text{ and } h_{ui} \neq \varnothing \text{ and } P(a^t|h_{ui}) \geq \delta \\ const & \text{otherwise} \end{cases}$$

Here, $r_{ui} \in \mathbb{N}$ is the number of observations of a^t in h_{ui}. The p_{ui} value expresses whether u will perform a target action on i or not. If h_{ui} contains actions of a target type, then $p_{ui} = 1$ with the confidence proportional to r_{ui}. For the non-empty histories without target actions, the $\mathcal{M}$ model estimates the probability $P(a^t|h_{ui})$ that the next unobserved action in h_{ui} will be of a target type. The probabilities enrich the sets of indicator and confidence values, thereby increasing the amount of information available to the WR-MF model. However, they also force AMMF to believe that it is likely to observe the target actions that have never happened. Such a bias can reduce the prediction quality of AMMF, especially if $\mathcal{M}$ is not able to capture well the correlation between actions of target and non-target types. Therefore, the threshold value $\delta \in [0, 1]$ is introduced. The higher the value of δ, the less strongly AMMF believes in the output of $\mathcal{M}$. When $\delta = 1$ the AMMF is equivalent to WR-MF.

The confidence value for $p_{ui} = 1$ is proportional either to the number of observations of the target actions in h_{ui} or to the estimated probability $P(a^t|h_{ui})$. Analogously to p_{ui}, we constrain the probability values with some threshold to mitigate the bias. The confidence value for $p_{ui} = 0$ is a constant that is usually much smaller than the confidence value for $p_{ui} = 1$.

4.2 Predicting a Target Action with a User-item Interaction History

Let us consider a user-item interaction history $h_{ui} = < a^1_{ui}, a^2_{ui}, \ldots, a^n_{ui} >$ where a^j_{ui} is the j^{th} action performed by u on i. We refer to this representation as a *timeless* history. *Time-aware* histories are instead represented as $< (a^1_{ui}, d^1_{ui}), \ldots, (a^n_{ui}, d^n_{ui}) >$, where d^j_{ui} is the observed time delay (e.g., in seconds) between action a^j_{ui} and a^{j+1}_{ui}. The last time delay d^n_{ui} is the time passed after the last action a^n_{ui} was performed by u on i.

A probabilistic model $\mathcal{M}$ predicts, for all the user-item histories that do not contain the target action a^t, the probability that the next unobserved user action will be of the target type. This task can be considered as a classification problem where $\mathcal{M}$ is a classifier and there are two possible outcomes; either the next unobserved user action will be of the target type (class 1) or not (class 0). Correspondingly, $P(a^t|h_{ui})$ is the probability to observe class 1.

To train a classifier to distinguish between two classes the examples of each of the classes have to be determined. Each user-item history can be split into two parts: *prefix* and *outcome*. The outcome represents an action in h_{ui}, while the prefix is a subsequence of h_{ui}, possibly empty, followed by the outcome. If h_{ui} contains a^t, when the outcome is the first a^t occurrence in h_{ui} and the prefix is a part of h_{ui} before the outcome. Otherwise, if h_{ui} does not contain a^t, the prefix is represented by all the actions in h_{ui} except the last one which is considered as the outcome. For example, let us consider the timeless user-item history $< 1,2,3,2 >$ where digits represent the action types. If $a^t = 3$, then the history is split into the prefix $< 1,2 >$ and the outcome 3. Differently, when $a^t = 4$, the history is split into the prefix $< 1,2,3 >$ and the outcome 2. The prefixes derived from h_{ui} containing a^t are used for predicting the class 1, while the prefixes derived from the histories that do not contain a^t are used for predicting the class 0.

The extracted prefixes are exploited to train the $\mathcal{M}$ model to predict the probabilities $P(a^t|h_{ui})$. Since the number of examples of each class may vary considerably, the class imbalance problem [3] has to be addressed. To combat it different strategies can be applied among which is the use of sampling methods [33].

Below we consider two types of classifiers which can be used as a $\mathcal{M}$ model. The classifiers of the first type do not take into account the ordering of the actions in user-item interaction histories. These classifiers make predictions by using features describing the corresponding user-item histories (e.g., the number of observations of actions of each type, the time of the last action, etc.). Conversely, the classifiers of the second type exploit the ordering of the actions and the time delays between them. These classifiers make predictions by using Sequence Mining (SM) techniques, such as Markov Chains.

4.2.1 History-summarizing Approaches

The history-summarizing approaches, which are described in this section, transform the user-item interaction history h_{ui} into the vector $\mathbf{x_{ui}} \in \mathbb{R}^m$ which represents with m features statistical information about the content of h_{ui}. For example, it may contain the information about the number of actions in h_{ui} or the time passed since the last action has been performed by u on i. This vector is then used by a classifier to estimate the probability that the next unobserved action in h_{ui} will be of the target type. Different classifiers, such as Logistic Regression, Support Vector Machines, or Decision Trees, can be applied for this task [6].

In this chapter, we consider the Logistic Regression (LR) classifier. LR estimates the probability to observe the action of the target type as follows:

$$P(a^t|h_{ui}) = P(a^t|\mathbf{x_{ui}}) = \frac{1}{1+e^{-(\mathbf{w}^T\mathbf{x_{ui}}+b)}}$$

The vector $\mathbf{w} \in \mathbb{R}^m$ and the scalar $b \in \mathbb{R}$ are parameters of LR learned during the classifier's training step. To fit the LR model, the training user-item histories are first split into prefixes and outcomes. The prefixes are transformed into the feature vectors, while the outcomes into the class labels (i.e., target and non-target). Using optimization techniques, such as gradient descent, we maximize the likelihood function on the transformed training data.

In general, ML algorithms do not perform well when the input numerical attributes, such as in the vector $\mathbf{x_{ui}}$, have different scales [28]. A good way to avoid this problem is to *standardize* the data so that all the attributes are given a mean of zero and a standard deviation of one [40]. That is why the attributes in $\mathbf{x_{ui}}$ are standardized before performing training and prediction.

4.2.2 Sequence-aware Approaches

In this section, we introduce several alternative sequence mining (SM) models which leverage information about the ordering of the actions and the time delays between them to predict the probability that the next unobserved user action in h_{ui} will be of the target type. During the training and prediction steps, a model-specific transformation function is applied to user-item histories. For instance, the function can convert the history $< 1,2,3 >$ into the set of suffixes $\{< 1,2,3 >, < 2,3 >, < 3 >, <>\}$. We refer to the output of the transformation function as a set of *keys*. In contrast to a feature vector, a key is a specialized representation of the corresponding history h_{ui} rather than statistical information about its content.

During the training step, user-item histories are split into prefixes and outcomes, which are in turn transformed into the keys and class labels respectively. Each key is mapped to

the number of times it has been observed with a particular class. During the prediction step, a user-item history is converted into a set of keys. Given the of set keys and the mapping described above, one can estimate the probability of the target class.

Transforming timeless histories into keys is quite straightforward; for example, histories can be transformed element-wise or by considering their bi-gram representations. However, time-aware histories require more complex approaches. In that respect, we suggest to discretize a continuous time delay into three levels: s - short, m - medium and l - long. For instance, the time-aware session $<(click,5),(bookmark,15),(reply,60)>$ is discretized as $<(click,s),(bookmark,m),(reply,l)>$. In general, the number of levels may vary depending on the data and the domain.

To simplify the forthcoming explanation of the models, we introduce here some convenient notations. We denote with $N(k)$ the function that returns the number of times the key k has been observed in the training set. The function $N_t(k)$ returns the number of times the key k has been observed together with the outcome a^t in the training set. Let us denote with N the total number of user-item histories in the training set. Analogously, N_t is the number of histories in the training set with the outcome a^t. The number of keys learned by a model is K.

SeqSet

This target action probability prediction model employs a transformation function T that converts an input history h_{ui} into a set of distinct actions and delays constituting h_{ui}. For example, T transforms a user-item history $<(1,s),(2,m),(2,m)>$ into a set $\{1,2,m,s\}$. The model estimates the probability that the next unobserved user action will be of the target type by employing the resulting set as the key k_{ui}:

$$k_{ui} = T(h_{ui})$$

$$P(a^t|h_{ui}) = P(a^t|k_{ui}) = \begin{cases} \frac{N_t(k_{ui})}{N(k_{ui})} & if\ N(k_{ui}) > 0 \\ \frac{1}{K} & otherwise \end{cases}$$

Bayes

The T function of the Bayes model transforms an input history h_{ui} into a list of suffixes arranged in descending order according to the size of a suffix. In the case of a time-aware

history, the transformation is defined as follows:

$$T(h_{ui}) = T(<(a^1_{ui}, d^1_{ui}), (a^2_{ui}, d^2_{ui}), \ldots, (a^n_{ui}, d^n_{ui})>) =$$
$$[<(a^1_{ui}, d^1_{ui}), (a^2_{ui}, d^2_{ui}), \ldots, (a^n_{ui}, d^n_{ui})>, <(a^2_{ui}, d^2_{ui}), \ldots, (a^n_{ui}, d^n_{ui})>, \ldots, <(a^n_{ui}, d^n_{ui})>, <>]$$

The idea behind such a transformation is that a user can perform the target action after any of the sub-histories of her interaction with a given item. During the training step, every suffix is used as a key to train the model. However, during the prediction step, only the longest suffix of h_{ui} presenting in the set of learned keys is employed. A similar procedure of shortening the initial sequence was proposed by Mobasher et al. [68].

Let us denote with l_{ui} the longest suffix of h_{ui} known to the prediction model. Then, the probability that the next unobserved user action will be of the target type can be estimated by applying Bayes' theorem:

$$P(a^t|h_{ui}) = P(a^t|l_{ui}) = \frac{P(l_{ui}|a^t)P(a^t)}{P(l_{ui}|a^t)P(a^t) + P(l_{ui}|a^{!t})P(a^{!t})}$$

$$P(a^t|l_{ui}) = \frac{\frac{N_t(l_{ui})}{N_t}\frac{N_t}{N}}{\frac{N_t(l_{ui})}{N_t}\frac{N_t}{N} + \frac{N_{!t}(l_{ui})}{N_{!t}}\frac{N_{!t}}{N}}$$

where $a^{!t}$ is an action not of the target type and $N_{!t}$ is the number of observations where the outcome is not of the target type of action.

Naive Bayes

The Naive Bayes (NB) model assumes that actions in a history are mutually independent, conditioned to the fact that the last action in h_{ui} is a^t (or $a^{!t}$ in the case of $P(h_{ui}|a^{!t})$). This is equivalent to assume that the input history is described by the list of its composing actions (action-delay pairs in the case of time-aware histories).

$$T(h_{ui}) = T(<(a^1_{ui}, d^1_{ui}), (a^2_{ui}, d^2_{ui}), \ldots, (a^n_{ui}, d^n_{ui})>) =$$
$$[<(a^1_{ui}, d^1_{ui})>, <(a^2_{ui}, d^2_{ui})>, \ldots, <(a^n_{ui}, d^n_{ui})>]$$

Let us denote with $h_{ui}[j]$ the j^{th} action (or action-delay pair) of h_{ui}. During the training step, every composing part is used as a key to train the model. The probability is estimated by applying Bayes' theorem:

$$P(a^t|h_{ui}) = \frac{P(h_{ui}|a^t)P(a^t)}{P(h_{ui}|a^t)P(a^t) + P(h_{ui}|a^{!t})P(a^{!t})}$$

Table 4.2: A sample of the interactions data

user_id	*item_id*	*interaction_type*	*created_at*
1974005	2668706	1	1444154047
2690450	405777	2	1445338496
2690450	1180447	3	1444806365
2690450	1981683	4	1444894156

However, because of actions independence we have:

$$P(h_{ui}|a^t) = P(h_{ui}[1]|a^t) \cdots P(h_{ui}[n]|a^t)$$

By applying smoothing we also have:

$$P(h_{ui}[j]|a^t) = \frac{N_t(h_{ui}[j]) + 1}{N(h_{ui}[j]) + K}$$

4.3 Datasets

In this section, we present the datasets used for the empirical evaluation of the proposed models. These are publicly available datasets from three different domains, namely job posting, movie, and retail. Each dataset contains user actions of more than two types.

4.3.1 XING Dataset

This dataset is provided by XING[1], a career-oriented social networking platform, for the ACM RecSys Challenge 2016. The dataset records interactions performed by users on job postings. A sample of the data is presented in Table 4.2. Actually, this dataset is a semi-synthetic sample of an original XING data, and it is enriched with artificial users whose presence contributes to the anonymization. Moreover, some noise was added to the data: not all interactions of a user are contained in the dataset while some of the interactions are artificial (have actually not been performed by the user)[2].

A row in the table represents an action/interaction of type *interaction_type* performed by the user *user_id* on the job posting (item) *item_id* at time *created_at*. There are four types of actions: 1 – the user *clicked* on the item, 2 – the user *bookmarked* the item, 3 –

[1]https://www.xing.com
[2]https://github.com/recsyschallenge/2016/blob/master/TrainingDataset.md

Table 4.3: Statistics of the XING data

# of users	784687
# of items	1029480
# of actions	8826678
% of click actions	81.4
% of bookmark actions	2.3
% of reply actions	4.8
% of delete actions	11.5

the user clicked on the *reply button* or *application form button* and 4 – the user *deleted* the recommendation from her recommendations list. Replies are usually selected as the target actions when working with this dataset. Table 4.3 shows statistics about the users, items (postings), and observations of different action types.

4.3.2 Rekko Dataset

Rekko dataset is provided by a Russian online cinema Okko[3] for the Rekko Challenge[4]. It comprises actions of multiple types performed by users while interacting with the online cinema. The dataset is an anonymized sample of user actions collected within an interval of more than 60 days. Timestamps are expressed in some abstract units, for which the distance and the order are preserved.

The actions are split into three categories: *transactions*, *ratings*, and *bookmarks*. Each transaction describes a video consumption (view) where the video is either available to a user through *subscription* or is *purchased* or *rented* by the user. Moreover, a transaction contains information about the amount of time a user watched a video, where the video is either a movie or a series. The goal of the challenge was to predict videos from the testing set, which was hidden from the challenge contestants, that a user *effectively* consumed. An effective consumption of a video is defined as follows:

- the user purchased or rented the video;
- the video is a movie available through subscription, and the user watched at least half of it;
- the video is a series available through subscription, and the user watched at least a third of it.

[3]https://okko.tv
[4]https://boosters.pro/championship/rekko_challenge

Table 4.4: Statistics of the Rekko data

# of users	500000
# of items	9580
# of actions	11030018
% of bookmarks	8.6
% of positive ratings	2.1
% of negative ratings	1.9
% of target transactions	59.2
% of non-target transactions	28.2

Table 4.5: Statistics of the Taobao data

# of users	98799
# of items	1589130
# of actions	10018144
% of click actions	89.5
% of purchase actions	5.6
% of add-to-cart actions	2.9
% of add-to-favorite actions	2.0

Thus, the transactions can be of two types. A *target* transaction describes an effective consumption of a video by a user. A *non-target* transaction is a short duration video view.

Taking into account the goals of the challenge, we transformed the original dataset into a set of interactions similar to those presented in Table 4.2. After the transformation, five types of user actions left: bookmark, positive rating, negative rating, target transaction, and non-target transaction. The *positive* (*negative*) ratings correspond to the rating observations which are greater or equal (lower) than the average rating in the dataset. Table 4.4 shows statistics about the users, items (videos), and observations of different action types.

4.3.3 Taobao Dataset

The last dataset is provided by Alimama [107, 108] and contains a sample of anonymized user behavior data from Taobao[5], a Chinese online shopping website. The dataset was collected during the period from 25.11.2017 to 03.12.2017 and contains user action of four types, namely click, purchase, add-to-cart, and add-to-favorite.

The original dataset is transformed into a form similar to one presented in Table 4.2. Statistics about the users, items, and observations of action types is presented in Table 4.5. We consider purchases as the target actions in our experiments.

[5]https://taobao.com

4.4 Evaluation Scenarios

To compare the previously presented models, we consider three evaluation scenarios. In the first scenario, the single action WR-MF model is compared with the multi action MMF and AMMF models detailed below. The purpose of this scenario is two-fold. First, to show that the use of multiple action types can be beneficial. Second, to check the hypothesis that the predictions provided by models leveraging ML techniques to discover the relationships between actions types automatically can be more accurate than those provided by heuristic-based models described in the previous chapter. In the second scenario, we compare the AMMF models employing history-summarizing approaches with those employing sequence-aware approaches. Both the first and the second scenarios are evaluated on the XING dataset (Section 4.3.1). We selected only one dataset since the process of designing the heuristic-based MMF model is difficult and requires careful analysis of the dataset. Using any AMMF model is easier than using the MMF model. In the third scenario, we compare AMMF with another multi action implicit feedback model called MF-BPR [61] on all three datasets introduced in Section 4.3.

Although in the experiments we use the data provided by companies for certain competitions, our goals are different from those selected for the competitions. That is also why we compare our models with state-of-the-art models rather than with the competition-winning ones. At the same time, it is worth noting that our AMMF model won the bronze medal of the Rekko Challenge.

4.4.1 Predicting Replies with Click and Bookmark Actions

In this section, we compare the single action WR-MF model with the multi action MMF and AMMF models on the XING dataset. The dataset contains user actions of four types (click, bookmark, reply, and delete) performed by users on job postings. Let us consider a scenario in which new replies on job postings should be predicted based on histories of users' clicks, bookmarks, and replies. We choose this scenario to simplify the creation of a heuristic-based MMF model. Nevertheless, it allows us to illustrate the difference in the methods of creating the models, their optimization, and the quality of prediction on the same set of action types.

WR-MF

The single action WR-MF model employing only information about reply actions has been selected as a baseline model for the evaluation scenario. The following indicator and

confidence functions have been used in the model:

$$p_{ui} = p(h_{ui}) = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

$$c_{ui} = c(h_{ui}) = \alpha r_{ui} + 1$$

Here, $r_{ui} \in \mathbb{N}$ is the number of observations of the reply action in the user-item interaction history h_{ui}. The target action prediction is computed as a dot product of f-dimensional user and item factors vector $x_u \in \mathbb{R}^f$ and $y_i \in \mathbb{R}^f$:

$$\hat{p}_{ui} = x_u^T y_i$$

The vectors' parameters are learned by minimizing the following objective function:

$$\min_{x*,y*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

Objective function minimization is achieved by alternating least squares (ALS) optimization [38, 76]. The optimal values for f, α, and λ are determined during the validation step.

MMF

To build the heuristic-based MMF model, we first performed exploratory data analysis of the XING dataset. According to it, both clicks and bookmarks positively correlate with replies. The proportion of user-item histories in which a reply action is following a click is 6%, a bookmark is 57%, and a click and a bookmark is 65%.[6] We used this information to design the MMF model presented in the following.

Let us denote with $l_{ui} \in \mathbb{N}$, $b_{ui} \in \mathbb{N}$, and $r_{ui} \in \mathbb{N}$, the number of clicks, bookmarks and replies observed in h_{ui}. Then, the indicator function for MMF can be defined as:

$$p_{ui} = p(h_{ui}) = \begin{cases} 1 & r_{ui} + l_{ui} + b_{ui} > 0 \\ 0 & r_{ui} + l_{ui} + b_{ui} = 0 \end{cases}$$

In other words, all three types of actions are provided to MMF as positive examples.

Similarly to WR-MF, if h_{ui} is empty, then we assume with a low confidence that $p_{ui} = 0$. Analogously, when h_{ui} contains replies the confidence should be proportional to r_{ui}. In

[6] There is no adjacency requirement between clicks/bookmarks and replies.

this case, there is no need to reinforce the confidence of $p_{ui} = 1$ with the information about clicks and bookmarks since this can lead to overfitting. Finally, the confidence value for a non-empty history without replies has to be proportional to l_{ui} and b_{ui}. The confidence that $p_{ui} = 1$ should be higher when the user clicked and bookmarked the item rather than just clicked or just bookmarked it. Thus, the confidence value can be defined as follows:

$$c_{ui} = c(h_{ui}) = \begin{cases} \alpha r_{ui} + 1 & r_{ui} > 0 \\ \beta b_{ui} + \gamma l_{ui} + 1 & r_{ui} = 0 \end{cases}$$

The target action prediction is computed using the same MF technique as in WR-MF. The optimal values for α, β, γ, and other hyper-parameters are determined during the validation step.

AMMF(LR-cb)

We recall that AMMF is a hybrid action prediction model which consists of two components: the $\mathcal{M}$ and WR-MF models. The $\mathcal{M}$ model predicts, for all the user-item histories that do not contain the target action a^t, the probability that the next unobserved user action will be of the target type. The WR-MF model, employs $\mathcal{M}$ and the collaboration between multiple users and items to produce the final action predictions.

To build the AMMF model, the $\mathcal{M}$ model has to be selected. In this scenario, we consider LR as the $\mathcal{M}$ model. The input to LR is a history-summarizing vector $\mathbf{x_{ui}}$ which, in our case, contains the number of clicks and bookmarks in the corresponding history h_{ui}. Hence, the probability to observe a^t for a given h_{ui} is:

$$P(a^t|h_{ui}) = P(a^t|\mathbf{x_{ui}}) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x_{ui}} + b)}} = \frac{1}{1 + e^{-(\mathbf{w}^T [l_{ui}, b_{ui}] + b)}}$$

where $\mathbf{w} \in \mathbb{R}^2$ and $b \in \mathbb{R}$ are parameters of LR learned during the classifier's training step. We refer to the AMMF(LR) model which employs the number of clicks and bookmarks as AMMF(LR-cb).

Accordingly, knowing $P(a^t|h_{ui})$ one can identify the indicator and confidence functions for AMMF. For this scenario, the indicator and confidence values can be identified as follows:

$$p_{ui} = p(h_{ui}) = \begin{cases} 1 & r_{ui} > 0 \\ 1 & r_{ui} = 0 \text{ and } h_{ui} \neq \varnothing \text{ and } P(a^t|h_{ui}) \geq \delta \\ 0 & \text{otherwise} \end{cases}$$

$$c_{ui} = c(h_{ui}) = \begin{cases} \alpha r_{ui} + 1 & r_{ui} > 0 \\ \beta P(a^t|h_{ui}) + 1 & r_{ui} = 0 \text{ and } h_{ui} \neq \varnothing \text{ and } P(a^t|h_{ui}) \geq \delta \\ 1 & \text{otherwise} \end{cases}$$

The target action prediction is computed using the same MF technique as in WR-MF. The optimal values for α, β, δ, and other hyper-parameters are determined during the validation step.

One may notice that $\mathcal{M}$ can be used as a stand-alone model predicting whether u will perform the target action on i or not. While it is true, such a model cannot benefit from CF. Hence, the quality of the predictions will be poor. On the other hand, it is conjectured that an AMMF is able to better identify correlations between actions of different types. Therefore, the predictions provided by AMMF should be more accurate than those provided by the heuristic-based MMF model.

4.4.2 Predicting Replies with All the Available Information

In the second evaluation scenario, which is also conducted on the XING dataset, new replies on job postings are predicted based on all the available historical information about users' interactions with items. In other words, the models are built using information about clicks, bookmarks, replies, deletes, and the timestamps when the user actions were performed. The scenario is created to illustrate the difference between the AMMF models employing history-summarizing approaches and those employing sequence-aware approaches. In particular, the AMMF(LR) model has been compared with the AMMF(NB) model. NB has been chosen as the sequence-aware $\mathcal{M}$ model since in our preliminary work it provided the best prediction accuracy.

AMMF(LR-all)

We use AMMF(LR), which has been introduced in Section 4.4.1, as the AMMF model employing a history-summarizing approach. In the current scenario, the feature vector $\mathbf{x_{ui}}$ describing the history h_{ui} contains information about the number of clicks, bookmarks, deletes, as well as the number of seconds passed since the last action in h_{ui}. The mentioned features are denoted with $l_{ui} \in \mathbb{N}$, $b_{ui} \in \mathbb{N}$, $d_{ui} \in \mathbb{N}$, and $\Delta_{ui} \in \mathbb{N}$, respectively. LR predicts, for all the user-item histories that do not contain the target action a^t, the probability that the next unobserved user action will be of the target type as:

$$P(a^t|h_{ui}) = P(a^t|\mathbf{x_{ui}}) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x_{ui}} + b)}} = \frac{1}{1 + e^{-(\mathbf{w}^T [l_{ui}, b_{ui}, d_{ui}, \Delta_{ui}] + b)}}$$

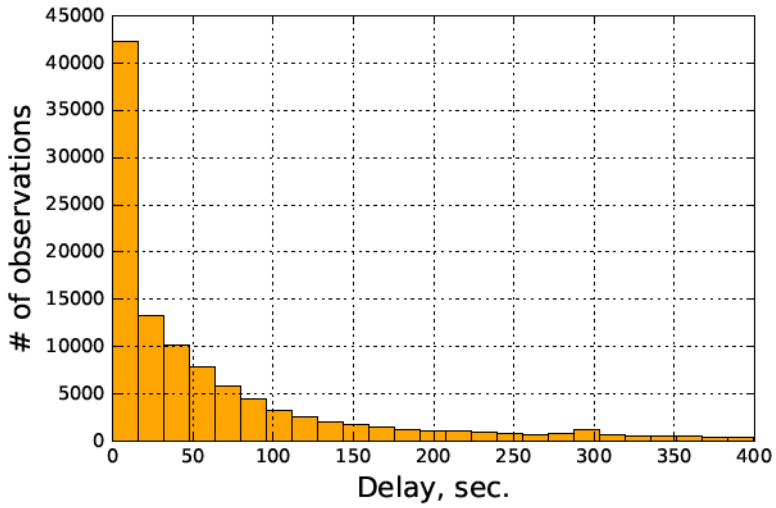

Figure 4.2: The distribution of the time delays between actions of all users. The largest delay is 400 seconds.

The vector $\mathbf{w} \in \mathbb{R}^4$ and the scalar $b \in \mathbb{R}$ are learned during LR's training step. The other key components of AMMF(LR), such as the indicator and confidence functions, remain unchanged. It is worth to note that the proposed AMMF model allows incorporating not only the information available in the actions of different types but also the information about the time. We refer to the AMMF(LR) model employing the number of clicks, bookmarks, deletes, and the number of seconds passed since the last action as AMMF(LR-all).

AMMF(NB)

In the AMMF(NB) model, NB is used as $\mathcal{M}$ model. We recall that NB is the sequence-aware approach, i.e., it exploits the ordering of the actions and the time delays between them to predict whether a user will perform an action of a target type on an item, given the user-item interaction history h_{ui}. The target action probability is estimated by applying Bayes' theorem with strong (naive) independence assumptions between the actions (action-delay pairs) composing h_{ui}.

To fit the NB model on time-aware user histories, the continuous time delays between actions have to be discretized. In this scenario, we discretize the time delays into three levels: s - short, m - medium and l - long. To identify the endpoints of the discrete intervals we analyzed the distribution of the time delays observed in all the available histories (Fig. 4.2). The 0.33 and 0.66 quantiles of this distribution have been used as separating values between the s, m and l intervals. Thus, the time delays lying in the intervals $[0, 0.33q)$ and $[0.33q, 0.66q)$ are considered as s and m, while time delays greater or equal to $0.66q$ are considered as l.

The estimated probability $P(a^t|h_{ui})$ is used to calculate the indicator and confidence values. Similarly to AMMF(LR), they are defined as:

$$p_{ui} = p(h_{ui}) = \begin{cases} 1 & r_{ui} > 0 \\ 1 & r_{ui} = 0 \text{ and } h_{ui} \neq \varnothing \text{ and } P(a^t|h_{ui}) \geq \delta \\ 0 & \text{otherwise} \end{cases}$$

$$c_{ui} = c(h_{ui}) = \begin{cases} \alpha r_{ui} + 1 & r_{ui} > 0 \\ \beta P(a^t|h_{ui}) + 1 & r_{ui} = 0 \text{ and } h_{ui} \neq \varnothing \text{ and } P(a^t|h_{ui}) \geq \delta \\ 1 & \text{otherwise} \end{cases}$$

The target action prediction is computed using the same MF technique as in WR-MF. The optimal values for α, β, δ, and other hyper-parameters are determined during the validation step.

4.4.3 Comparing Multi Action Prediction Models

In the last evaluation scenario, AMMF(LR) has been compared with MF-BPR [61] on the datasets presented in Section 4.3. For each dataset, the models predict the target actions employing observations of all the action types available in the dataset and neglecting the time when the actions were performed. We limit the use of the time information since MF-BPR is not a time-aware model. The AMMF(LR) model, which uses only the number of observations of predictive action types, is denoted as AMMF(LR-nt).

Let us assume that a dataset contains user actions of $k+1$ types (i.e., k predictive and one target action types). In the case of AMMF(LR-nt), the feature vector $\mathbf{x_{ui}} \in \mathbb{N}^k$ corresponding to the user-item history h_{ui} contains information about the number of observations of predictive action types in h_{ui}. For all the user-item histories that do not contain the target action a^t, LR estimates the probability that the next unobserved user action will be of the target type. The parameters of the LR model are learned during the training step. The indicator and confidence functions for AMMF(LR-nt) remain unchanged.

In the case of MF-BPR, each action type is associated with a preference level. The optimal relative ordering of the levels is selected manually. The target action type has the highest preference level. Table 4.6 shows the datasets' action types sorted in ascending order of the preference level. Since in the training phase, in addition to the observed user actions, the model samples unobservable user-item pairs, we have introduced an additional type of "action" which we refer to as *unobserved*. The unobserved actions have higher preference

Table 4.6: Action types sorted in ascending order of the preference level

Dataset	Levels
XING	delete, unobserved, click, bookmark, **reply**
Rekko	negative rating, unobserved, non-target transaction, bookmark, positive rating, **target transaction**
Taobao	unobserved, click, add-to-favorite, add-to-cart, **purchase**

levels than negative ones. For example, in the case of Rekko, it is better to recommend to a user a movie that she had never watched before than the movie she rated negatively.

Let $\mathbb{L} = (L_1, ..., L_n)$ represent an ordered set of levels in a dataset such that a feedback in L_i is a stronger signal of interest compared to a feedback in L_{i-1}, that is $L_i > L_{i-1}$. A set of positive feedback levels, i.e. the levels which are higher than the layer related to the unobserved action type, is defined as $\mathbb{L}^+$. Thus, the training set for the MF-BPR model can be defined as follows:

$$D = \{(u, i, j) \mid i \in I_{L_a,u} \wedge j \in I_{L_b,u} \wedge L_a \in \mathbb{L}^+ \wedge L_b \in \mathbb{L} \wedge L_a > L_b\}$$

where the tuple (u, i, j) shows that user u prefers item i to item j, L_a and L_b are two levels, and $I_{L_k,u}$ is the set of items in level L_k that u interacted with. The tuples are sampled based on the given levels. We use the sampling techniques proposed by Loni et al. [57] for the MF-BPR-ML model, which, according to the cited paper, is the best performing variation of MF-BPR. For the same reason, in our experiments, level L_b corresponds only to unobserved actions (MF-BPR's hyper-parameter $\beta = 1$).

The AMMF(LR-nt) and MF-BPR models are trained using the ALS and Stochastic Gradient Descent (SGD) optimization techniques, respectively.

4.5 Evaluation

This section presents the evaluation protocol, metrics, and results for the three scenarios mentioned above. First, we describe how the training and testing sets are created. Then, we present the offline metrics which we use to compare the models from the evaluation scenarios. Finally, for each scenario, we present the results of the performed empirical evaluation.

4.5.1 Training and Testing Sets

The original data has been split into the training and testing sets using a time-dependent splitting criterion. The first 80% timestamped data records were used for the training, while

the remaining 20% was used as the testing set. Being split by a time point, a user-item history may fall partially in the training and partially in the test. In this case, the first part of the history is considered as a part of the training set, while the remaining part stays in the testing set. Only the target actions are kept in the testing set.

To optimize the hyper-parameters of the proposed models, we generated five training-validation sets from the original training set. Let us assume that the training set contains actions which have been observed within some interval $\mathcal{I}$. To generate a training-validation set we collect user actions from a random subinterval $\mathcal{I}'$ of $\mathcal{I}$. The data from $\mathcal{I}'$ is then split into the training (80%) and validation (20%) parts. The hyper-parameters were optimized by benchmarking the trained model on the validation parts. As soon as the hyper-parameters were identified, the models were retrained on the original training set. The models have never used the testing set throughout the training phase.

During the training phase of the $\mathcal{M}$ models (i.e., LR and NB), each training history is split into the prefix and outcome (see Section 4.2). The prefix-outcome pairs are partitioned into two groups: those whose outcome is of a target type (a^t) and those whose outcome is not of a target type ($a^{!t}$). We apply the random undersampling technique [33] to balance the resulting groups. The $\mathcal{M}$ models are trained on the data from the balanced groups.

To produce the final action predictions, most of the presented models use the MF technique which requires the learning of user and item latent feature vectors. To ensure sufficient training, without losing a large amount of the initial data, we kept in the dataset only users who performed target actions on at least two items and items on which at least two users performed target actions in the training set. Additionally, we removed from the testing set users and items that were not present in the training set. Finally, for each user, we removed the "easy" predictions from the testing set corresponding to the items on which that user performed target actions in the training set.

4.5.2 Protocol and Metrics

During the evaluation, for each user in the testing set a model generates a list of items (recommendations) on which it is predicted that the user will perform a target action. The items are then sorted by the predicted indicator score and the top-k of them are selected. The quality of the models has been assessed by measuring Mean Average Precision at k (MAP@k) and F1-score at k (F1@k) [63]. These metrics are based on the precision and recall metrics that we define below.

Similarly to Section 3.3, we denote with U^T the set of test users, with M_u the set of top-k ranked recommended items for a user u, and with T_u the set of items in the test data on which the user u performed the target action. Accordingly, Precision at k (P@k) and Recall at k

Table 4.7: Parameter values on which the search was carried out

Model	Hyper-parameters	Values
MF-based models	f, α, β, γ	20, 40, 60, 80, 100, 120, 140, 160, 180, 200
	δ	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
	λ	0.1, 0.01, 0.001, 0.0001
LR	C	1.0, 0.1, 0.01, 0.001
	penalty	$l1, l2$

(R@k) are defined as:

$$P@k = \frac{1}{|U^T|} \sum_{u \in U^T} \frac{|M_u \cap T_u|}{k}$$

$$R@k = \frac{1}{|U^T|} \sum_{u \in U^T} \frac{|M_u \cap T_u|}{|T_u|}$$

MAP is the arithmetic mean of average precision values across recall levels over test users. If J_u is the set of the positions of the relevant items for a user u in M_u, then:

$$MAP@k = \frac{1}{|U^T|} \sum_{u}^{U^T} \frac{1}{|J_u|} \sum_{j}^{J_u} P@j$$

F1-score is the harmonic mean of precision and recall, thus:

$$F1@k = \frac{2 * P@k * R@k}{P@k + R@k}$$

In the evaluation experiments $k \in \{1,3,5,7,10\}$, since in real-world applications only the very first recommended items are typically browsed by users.

4.5.3 Comparing Single and Multi Action Prediction Models

The evaluation scenario formulated in Section 4.4.1 assumes the comparison of four models: WR-MF, MMF, AMMF(LR-cb), and the stand-alone LR model. The models' hyper-parameters have been tuned by optimizing the prediction accuracy of the models on the validation set in terms of F1@5. The randomized search [25] method for parameter optimization has been applied. Parameter values on which the search was carried out are shown in Table 4.7. The optimal values for models' hyper-parameters are presented in Table 4.8.

Table 4.8: The models' hyper-parameters and their optimal values

Model	Hyper-parameters
WR-MF	$f = 100$, $\lambda = 0.1$, $\alpha = 100$
MMF	$f = 180$, $\lambda = 0.1$, $\alpha = 80$, $\beta = 100$, $\gamma = 120$
AMMF(LR-cb)	$f = 160$, $\lambda = 0.1$, $\alpha = 200$, $\beta = 160$, $\delta = 0.8$
LR	$C = 1$, $penalty = l1$

Table 4.9: Comparison of models in terms of MAP@k and F1@k. Best values are underlined.

Model	MAP					F1				
	@1	@3	@5	@7	@10	@1	@3	@5	@7	@10
LR	0.027	0.030	0.031	0.027	0.021	0.030	0.034	0.033	0.030	0.025
WR-MF	0.022	0.039	0.045	0.047	0.049	0.026	0.040	0.039	0.035	0.031
MMF	0.025	0.043	0.051	0.054	0.059	0.028	0.040	0.040	0.038	0.037
AMMF(LR-cb)	0.027	0.043	0.052	0.056	0.060	0.030	0.040	0.047	0.045	0.040

For LR, the implementation provided by `scikit-learn`[7] package has been used. The parameter *penalty* specifies the regularization norm, while $C \in \mathbb{R}^+$ is the inverse of regularization strength. Smaller values of C provide stronger regularization.

The trained LR classifier is also used as the stand-alone model. Since all the "easy" predictions have been removed from the testing set, the model predicts the probability $P(a^t|h_{ui})$ that in an interaction history h_{ui}, which does not contain the target action, the next performed by the user action will be of a target type. The estimated probability is then used as the predicted indicator score.

In MF, the user and item latent feature vectors are initialized randomly. This may lead to the situation when the order and content of the recommendations provided by the model vary from training to training. To mitigate this problem, the final score for each metric is calculated as an average over five consecutive evaluations of each model. The evaluation results are reported in Table 4.9.

As it was expected, the worst model in terms of both MAP@k and F1@k is the stand-alone LR model. The reason is that this model considers each user-item history independently and does not benefit from CF. The MMF model almost always outperforms the WR-MF model, which is particularly noticeable in terms of MAP@k. We assume that this improvement is associated with the better prediction precision, which is in turn associated with the increased number of non-zero indicator scores in MMF. This also explains why with MAP@1 and F1@1 the MMF model performs better: when there is only one slot for recommendations the

[7] http://scikit-learn.org

precision plays a bigger role than the recall. The number of non-zero elements in the user-item indicator matrices and their densities for each MF model are presented in Table 4.10.

Table 4.10: Statistics of the indicator matrices

Model	Density	# of non-zero cells
WR-MF	0.00014	95870
MMF	0.00042	279477
AMMF(LR-cb)	0.00019	128444

In contrast to MMF, the AMMF(LR-cb) model, while having almost the same number of indicator scores as WR-MF, performs better than other presented models. It improves both the prediction recall and precision.

The empirical evaluation has shown that the proposed multi action models outperform the model employing only the target action data. It has also shown that the AMMF model employing LR to discover relationships between actions of different types automatically can outperform the heuristic-based MMF.

4.5.4 Comparing History-summarizing and Sequence-aware Approaches for AMMF

In the second evaluation scenario, which is described in Section 4.4.3, AMMF(LR-all) is compared to AMMF(NB). We have used the same procedure as in the previous section to fine-tune the models' hyper-parameters. The optimal values for hyper-parameters are presented in Table 4.11.

The models have been compared in terms of MAP@k and F1@k (Table 4.12). The optimal number of training iterations for the AMMF(LR-all) and AMMF(NB) models is 15. The final score for each metric is calculated as an average over the five consecutive evaluations of each model. For clarity, we show again here the values of the metrics for the WR-MF and AMMF(LR-cb) models.

Similarly to the previous scenario, the models employing actions of multiple types outperform the single action WR-MF model. The AMMF(LR-all) model incorporating all the available information performs the best, while AMMF(LR-cb) outperforms AMMF(NB).

Table 4.11: The models' hyper-parameters and their optimal values

Model	Hyper-parameters
AMMF(LR-all)	$f = 140$, $\lambda = 0.1$, $\alpha = 80$, $\beta = 180$, $\delta = 0.1$
AMMF(NB)	$f = 120$, $\lambda = 0.01$, $\alpha = 140$, $\beta = 80$, $\delta = 0.8$

Table 4.12: Comparison of models in terms of MAP@k and F1@k. Best values are underlined.

Model	MAP					F1				
	@1	@3	@5	@7	@10	@1	@3	@5	@7	@10
WR-MF	0.022	0.039	0.045	0.047	0.049	0.026	0.040	0.039	0.035	0.031
AMMF(LR-cb)	0.027	0.043	0.052	0.056	0.060	0.030	0.040	0.047	0.045	0.040
AMMF(LR-all)	0.027	0.049	0.056	0.060	0.063	0.031	0.052	0.049	0.045	0.040
AMMF(NB)	0.024	0.041	0.047	0.050	0.052	0.029	0.042	0.040	0.037	0.033

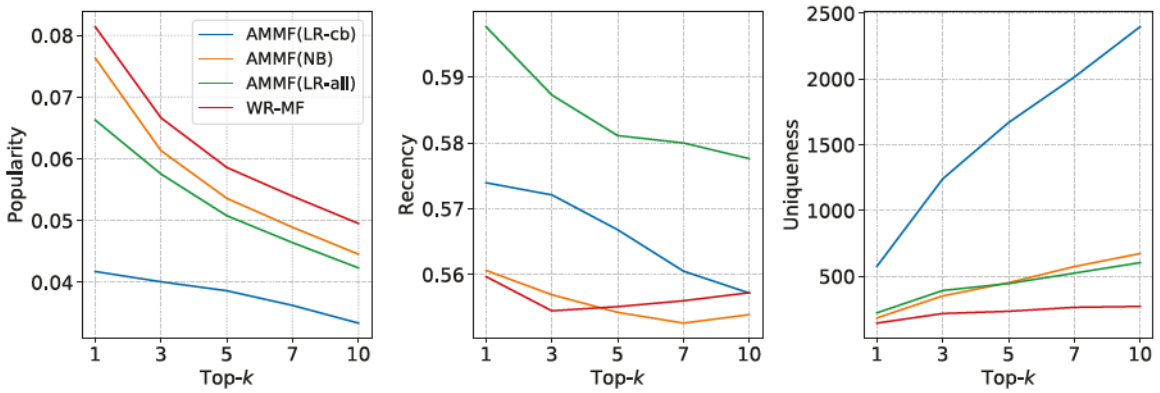

Figure 4.3: The popularity, recency, and uniqueness of the top-k items recommended by the models

To better understand the pros and cons of the proposed algorithms, we analyzed the recommendations provided by the models in terms of *popularity*, *recency*, and *uniqueness* (Fig. 4.3). We define the popularity of an item as the number of times it has been observed in the training set. The recency of an item is a value that is inversely proportional to the average amount of time passed since the interactions with this item. We also introduce uniqueness as the number of items recommended only by the particular model and not by the others. Both the recency and popularity metrics are normalized to the $[0,1]$ interval. The closer the popularity to 0, the less popular is the recommended item. The closer the recency to 1, the more recently users have interacted with the item.

Fig. 4.3 shows that the AMMF models recommend less popular items than WR-MF. The immunity towards popularity bias is traditionally considered as a good feature of a recommendation model. The least popular and the most unique items are recommended by the AMMF(LR-cb) model, while AMMF(LR-all) recommends more recent items than other models. The information about additional types of actions and time allows AMMF(NB) and AMMF(LR-all) to identify similar sets of item candidates for recommendations for a given moment of time. However, since the models have been optimized for regression rather than

Table 4.13: AMMF(LR-nt) and MF-BPR parameter values on which the search was taken. The "Common" row contains hyper-parameters related to both models.

Model	Hyper-parameters	Values
Common	f λ	20, 40, 60, 80, 100, 120, 140, 160, 180, 200 0.1, 0.01, 0.001
AMMF(LR-nt)	α, β δ	20, 40, 60, 80, 100, 120, 140, 160, 180, 200 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9
MF-BPR	γ	0.1, 0.01, 0.001

Table 4.14: The AMMF(LR-nt) and MF-BPR models' hyper-parameters and their optimal values for the XING, Rekko, and Taobao datasets

Model	Dataset	Hyper-parameters
AMMF(LR-nt)	XING	$f = 180$, $\lambda = 0.1$, $\alpha = 180$, $\beta = 160$, $\delta = 0.7$, $n = 15$
	Rekko	$f = 80$, $\lambda = 0.1$, $\alpha = 5$, $\beta = 120$, $\delta = 0.5$, $n = 15$
	Taobao	$f = 160$, $\lambda = 0.1$, $\alpha = 80$, $\beta = 140$, $\delta = 0.5$, $n = 20$
MF-BPR	XING	$f = 160$, $\lambda = 0.001$, $\gamma = 0.01$, $s = 8000$, $n = 100$
	Rekko	$f = 40$, $\lambda = 0.01$, $\gamma = 0.001$, $s = 10000$, $n = 100$
	Taobao	$f = 160$, $\lambda = 0.001$, $\gamma = 0.01$, $s = 4000$, $n = 70$

for ranking, all the good candidates may have very similar predicted scores which in turn can influence the selection of top-k items and, as a consequence, the scores of MAP@k and F1@k. Thus, we assume that by optimizing AMMF(LR-all) and AMMF(NB) for a ranking criterion, such as BPR-Opt [84], one can improve further the MAP@k and F1@k scores for these two models. We would like to validate this hypothesis in our future works.

4.5.5 Comparing Multi Action Prediction Models

In the last evaluation scenario, we compare AMMF(LR-nt) with MF-BPR [61] using all the datasets presented in Section 4.3. The models employ information about the actions of all the available types to predict whether a user will perform a target action on an item. The models' prediction accuracy has been assessed in terms of MAP@k and F1@k where $k \in \{1, 3, 5, 7, 10\}$.

To fine-tune the models' hyper-parameters, we have used the same procedure as in the previous sections. Parameter values on which the randomized search [25] was carried out are shown in Table 4.13. Here γ corresponds to the learning rate of the MF-BPR model. The sets of values are the same for all the datasets. The optimal values for AMMF(LR-nt)'s and MF-BPR's hyper-parameters are presented in Table 4.14. Parameter n denotes the optimal number of iterations needed to train the model, while s describes the number of tuples sampled for a training iteration of the MF-BPR model.

Table 4.15: The comparison of AMMF(LR-nt) and MF-BPR in terms of MAP@k and F1@k on the XING, Rekko, and Taobao datasets. Best values are underlined.

		XING		Rekko		Taobao	
		AMMF (LR-nt)	MF-BPR	AMMF (LR-nt)	MF-BPR	AMMF (LR-nt)	MF-BPR
MAP	@1	<u>0.027</u>	0.023	<u>0.013</u>	0.005	0.026	<u>0.035</u>
	@3	<u>0.044</u>	0.036	<u>0.022</u>	0.009	0.045	<u>0.051</u>
	@5	<u>0.051</u>	0.041	<u>0.026</u>	0.012	0.053	<u>0.056</u>
	@7	<u>0.055</u>	0.043	<u>0.029</u>	0.013	0.057	<u>0.058</u>
	@10	<u>0.059</u>	0.045	<u>0.031</u>	0.015	<u>0.060</u>	0.059
F1	@1	<u>0.029</u>	0.026	<u>0.021</u>	0.008	0.028	<u>0.037</u>
	@3	<u>0.040</u>	0.036	<u>0.037</u>	0.018	<u>0.040</u>	<u>0.040</u>
	@5	<u>0.039</u>	0.033	<u>0.044</u>	0.023	<u>0.039</u>	0.035
	@7	<u>0.037</u>	0.029	<u>0.047</u>	0.026	<u>0.038</u>	0.029
	@10	<u>0.036</u>	0.025	<u>0.048</u>	0.029	<u>0.033</u>	0.023

The results of the offline experiments conducted on three datasets are presented in Table 4.15. As one can see, in the case of the XING and Rekko datasets, AMMF(LR-nt) is superior to MF-BPR in terms of MAP@k and F1@k for all values of k. At the same time, in the case of Taobao, AMMF(LR-nt) outperforms MF-BPR in terms of F1@k but falls behind in terms of MAP@k. The performance difference is statistically significant with $p < 0.05$ (one-sided t-test) for MAP@$\{1,3,5\}$ and F1@$\{1,5,7,10\}$. The superiority of MF-BPR in terms of MAP@$\{1,3,5\}$ and F1@1 on the Taobao dataset can be explained by the fact that MF-BPR is optimized for ranking; having a smaller recall it still able to rank the relevant items higher. However, in this case, it is unclear why AMMF(LR-nt) outperforms MF-BPR in terms of MAP on other datasets. To understand the reason for such results, we performed a statistical analysis of the training parts of the datasets.

According to the statistics presented in Table 4.16, the XING and Taobao datasets are similar to each other in terms of the way users interact with items. On average, a user performs 1.87 actions on an item in XING versus 1.68 in Taobao. The average number of interactions for a user in XING and Taobao is 27.79 and 18.74, while for an item it is 21.47 and 20.88. At the same time, in both datasets, 25% of the user-item pairs associate with at least two interactions. On the other hand, differently from the XING dataset, which also contains user actions corresponding to negative feedback (delete), in Taobao, all the observed user actions correspond to positive implicit feedback. Therefore, in the case of Taobao, the MF-BPR model simply prioritizes items with which users interact. When the relations between the action types become more complex, as in the case of the XING and Rekko datasets, the AMMF(LR-nt) model performs better.

Table 4.16: The statistical analysis of the XING, Rekko, and Taobao datasets for the comparison of the AMMF(LR-nt) and MF-BPR models

	XING	Rekko	Taobao
# of users	22614	397220	10886
# of items	29280	7840	9769
# of unique user-item pairs	336955	7906316	121669
# of interactions for a user (mean)	27.79	21.14	18.74
# of interactions for an item (mean)	21.47	1071.31	20.88
# of interactions for a user-item pair (mean)	1.87	1.06	1.68
# of interactions for a user-item pair (p75)	2	1	2
# of interactions for a user-item pair (max)	53	3	82
% of target action types	21%	61%	15%
% of positive action types	90%	92%	100%

The evaluation results illustrate that the AMMF(LR-nt) model can better recognize the relationships between different types of user actions and employ them to provide better predictions. At the same time, in terms of ranking metrics, the model may be inferior to analogous models optimized for ranking. That is why, in our future works, we would like to optimize AMMF(LR) for a ranking criterion, such as BPR-Opt [84].

4.6 AMMF in Action

In many practical recommendation applications, the designer of the system aims at influencing user behavior. The real effect of the RS depends on a variety of factors such as the user's information needs, how much they trust the system, and the interface through which the recommendations are presented [29]. Unfortunately, it might be difficult or even impossible to take into account all these factors in an offline evaluation. That is why it is important to compare RSs online, where users that perform real tasks use the systems. In this section, we present an online evaluation of multi action prediction models that was conducted in an online testing system of a leading Russian video streaming service called *ivi*[8].

4.6.1 Introduction to *ivi*

To simplify the upcoming presentation of the online experiment, we first introduce *ivi*. *ivi* is an online video streaming service attracting more than 40 million unique visitors per month (including more than 1.5 million subscribers) and offering approximately 80000 titles of

[8]https://www.ivi.tv/

movies, series, cartoons, and TV programs. The service is available on all the major platforms, including web, smart devices (phones, tablets, TVs), and video gaming consoles (Xbox and PlayStation). Unlike Netflix, which operates on a subscription monetization model (i.e., if a user has purchased a subscription, then the entire video catalog is available to her), *ivi* relies on three different monetization models:

- advertising model - a user can get access to a small part of the catalog for free by watching advertisements;
- subscription model - a user can get an ad-free access to most of the video catalog by paying a subscription fee;
- transactional model - some recently released or cult films can be watched only if the user pays for each of it separately.

The presence of the transactional model is due to the requirements of copyright holders. Although the transactional model reduces the value of the subscription in the eyes of the user, it allows *ivi* to offer new films almost simultaneously with their release.

Like many other services employing RSs, *ivi* logs information about actions generated by users. In particular, it collects information about a user's *impressions* (items the user has seen on the service over the past few weeks), *clicks* (items the user clicked on), *video views* (items the user watched), *purchases* (items the user bought), and *ratings* (items the user rated). Some of this information is used to build recommendations. It is also worth noting that the number of actions varies from one action type to another and is not equal. For example, the number of video view is hundreds of times the number of ratings.

The *ivi* RS is used to generate the main page of the application, which consists of blocks (Fig. 4.4). Each block contains a set of items. Some blocks, such as the Central Promo Block, are fixed on particular positions. Others, such as collections, may change their position depending on their utility for the user, which is the probability that an item will be watched from the block. The larger the estimated utility of the block is, the higher is the block on the page.

The utility of a block is determined by the utility of items contained in it. The utility of an item is a combination of the score predicted by the recommender model and additional scores derived from business rules. The rules, such as calibration [96], are designed to increase the novelty and diversity of items. For each item, the recommender model predicts whether the user will watch this item or not. Thus, the predicted item utility scores are used to sort items within the blocks and the blocks on the page.

The quality of the RS is assessed by a number of business metrics, among which the main ones are the conversion into video view, item purchase, and subscription purchase. It

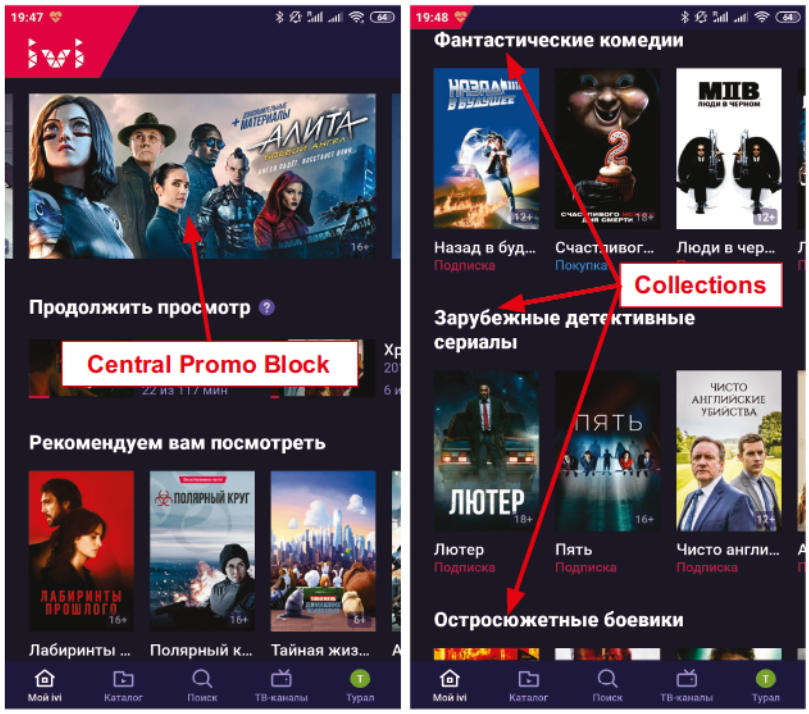

Figure 4.4: The main page of the *ivi* application for Android-based smartphones. The page consists of blocks, such as Central Promo Block (CPB) or item collections.

is also assumed that the RS may indirectly affect bounce and subscription churn rates [20] since the user is always exposed to such relevant items that she does not want to leave the service. All these metrics quite clearly correlate with the business objectives of the company.

At the time of the experiment, a heuristic-based MMF model, similar to those presented in Chapters 3, was already implemented in *ivi* by the company's engineers. The multi action prediction model, which we refer to as iviMMF, employs historical information about two types of actions, namely video view and rating, to predict whether a user will watch an item. The model operates in the real-time mode and rebuilds recommendations each time a user performs one of the predictive actions, i.e., either watches or rates an item. The online updates of the model are built on top of the fold-in technology proposed by Rendle et al. [86]. The complete offline retraining of the model takes place once a day.

Although iviMMF provides predictions by using video views and ratings, it does not take into account information available in other types of actions, such as impressions or clicks. For a long time, *ivi* engineers have tried to expand the model by searching for new heuristics to calculate the indicator and confidence values. However, all their attempts did not lead to success. That is why we decided to try the models that can automatically determine the relationship between the target and predictive action types. Due to requirements related to

response time and the necessity to support online updates of the model the list of candidate model has been reduced to two, namely AMMF(LR) and MF-BPR [61].

4.6.2 Prediction Models

In this section, we briefly present the models chosen as candidates for the online experiment. All the model are optimized to predict whether a user will watch an item.

iviMMF

iviMMF is a multi action prediction model developed by *ivi* engineers. The model generates recommendations by using information about video views and ratings. The indicator and confidence functions, which are based on heuristics determined by exploratory analysis of the data, are defined as follows:

$$p_{ui} = \begin{cases} 1 & r_{ui} > \bar{r}_u \\ 1 & v_{ui} > 0 \text{ and } \nexists r_{ui} \\ 0 & \text{otherwise} \end{cases}$$

$$c_{ui} = \begin{cases} \alpha * log(d_i + 1) + 1 & \exists r_{ui} \\ \alpha * log(v_{ui} + 1) + 1 & v_{ui} > 0 \text{ and } \nexists r_{ui} \\ 1 & \text{otherwise} \end{cases}$$

Here r_{ui} is the rating given by user u to item i, $\bar{r}_u$ is the average rating of u, v_{ui} is the total amount of time in seconds u watched i, d_i is the duration of i in seconds, and α is the hyper-parameter which is optimized during the cross-validation.

The motivation behind the selection of such indicator and confidence functions is the following. If a user watched an item, the target action was observed. Moreover, it can be conjectured that if a user rated an item, then most likely she watched it. At the same time, one would like to deprioritize items that the user watched but did not like. Therefore, if the user rated an item positively (the rating is greater than the user's average rating) or watched the item without rating it, then we assume that $p_{ui} = 1$, otherwise $p_{ui} = 0$.

Similarly to Hu et al. [38], the authors of the model hypothesize that the more a user watches the item, the higher is the confidence that the target action is observed. Since the duration of the item may vary, the confidence function is proportional to the logarithm of the duration of the video view. The use of the logarithm can help to mitigate the model's bias towards the prediction of long popular items. In case, when the user did not watch the item

but rated it, the authors assume that she watched the item till the end. Thus, in iviMMF, the rating action has a priority over the video view action.

AMMF(LR)

AMMF(LR) is multi action prediction model that we propose as a candidate for the online evaluation. The model makes predictions by using the information available in video views, ratings, clicks, impressions and purchases. Similarly to iviMMF, we designed the model to recommend items which a user would like to watch entirely and which the user would like to rate with a positive rating. The indicator and confidence functions for AMMF(LR) are defined as follows:

$$p_{ui} = p(h_{ui}) = \begin{cases} 1 & r_{ui} > \bar{r}_u \\ 1 & v_{ui} > 0 \text{ and } r_{ui} \notin h_{ui} \\ 1 & v_{ui} = 0 \text{ and } r_{ui} \notin h_{ui} \text{ and } h_{ui} \neq \varnothing \text{ and } P(a^t|h_{ui}) \geq \delta \\ 0 & \text{otherwise} \end{cases}$$

$$c_{ui} = c(h_{ui}) = \begin{cases} \alpha * log(v_{ui}+1)+1 & r_{ui} > \bar{r}_u \text{ and } v_{ui} > 0 \\ \alpha * log(\bar{d}_i+1)+1 & r_{ui} > \bar{r}_u \text{ and } v_{ui} = 0 \\ \alpha * log(v_{ui}+1)+1 & r_{ui} \notin h_{ui} \text{ and } v_{ui} > 0 \\ \beta * log(\bar{d}_i+1)+1 & r_{ui} \notin h_{ui} \text{ and } v_{ui} = 0 \text{ and } h_{ui} \neq \varnothing \text{ and } P(a^t|h_{ui}) \geq \delta \\ \gamma * log(v_{ui}+1)+1 & r_{ui} \leqslant \bar{r}_u \text{ and } v_{ui} > 0 \\ \gamma * log(\bar{d}_i+1)+1 & r_{ui} \leqslant \bar{r}_u \text{ and } v_{ui} = 0 \\ 1 & \text{otherwise} \end{cases}$$

Here h_{ui} is the user-item interaction history, r_{ui} is the rating given by u to i, $\bar{r}_u$ is the average rating of u, v_{ui} is the total amount of time in seconds u watched i, and $\bar{d}_i$ is the average amount of seconds spent by users watching i. In the case, when the user u interacted with the item i, but did not watch or rate it, we use a classifier predicting the probability that u will perform a target action a^t on i given the user-item interaction history h_{ui}. We denote this probability as $P(a^t|h_{ui})$. Finally, α, β, γ and δ are the hyper-parameter which are optimized during the cross-validation.

We use Logistic Regression (LR) to estimate the probability of observing the target action a^t given h_{ui}. The input to LR is a history-summarizing vector $\mathbf{x_{ui}} \in \mathbb{R}^3$ which, in our case, contains the number of clicks, impressions, and purchases in the corresponding history h_{ui}.

MF-BPR

MF-BPR is another candidate model for the online evaluation. In this model each type of action corresponds to a particular priority level. The model is trained to rank items for a user in such a way that the top recommended items are expected to get the most priority actions from the user. To train MF-BPR, we use the same action types as for AMMF(LR). The list of action types ordered in the ascending order of the priority looks as: negative rating, impression, unobserved action (see Section 4.4.3), click, purchase, video view, positive rating. Thus, it is assumed that the model will prioritize those items which the user will either watch or rate positively.

For the evaluation, we chose the MF-BPR-ML variation of MF-BPR [61], since, according to the original paper, it performs the best.

4.6.3 Offline Evaluation

The online experiment can have a negative effect on both the RS performance and the user experience. Therefore, to minimize the risks of the failure, it is best to run an online evaluation last, after conducting an extensive offline evaluation of the candidate models [29]. That is why we first present the result of an offline evaluation of the models introduced above.

For the offline evaluation, we used the data collected on *ivi* for the 30 days. The data from the first 25 days was used for the training, while the video views from the last five days were kept as the testing set. We applied cross-validation to identify the optimal values for models' hyper-parameters. For this reason, by using the sliding window, we split the training set into five subsets where each subset contains data from 21 consecutive days. The data from the first 20 days were used for training, while the last day was used for the validation.

Traditionally, in the movie domain, recommendations do not contain items that a user has already watched [38]. That is why we removed from the recommendations those items that have been previously watched or rated by the user. Moreover, during the validation and testing phases, the items and users that did not present in the training set were removed from the validation and test sets, respectively.

During the offline evaluation, the quality of the models has been assessed in terms of MAP@k and F1@k (see Section 4.5.2). For the experiments, we used the offline testing system designed by *ivi*. The system accepts only a limited set of values for k, that is why in our experiments $k \in \{1, 5, 10\}$.

The models' hyper-parameters have been tuned by optimizing the prediction accuracy of the models on the validation set in terms of F1@5. The grid search [25] method for parameter optimization has been applied. Parameter values on which the search was carried out are

Table 4.17: The hyper-parameters' values for grid search

Model	Hyper-parameter	Values
iviMMF	f	40, 50, 60
	α	1, 2, 5, 10, 20, 40, 60
	λ	0.1, 0.01, 0.001
AMMF(LR)	f	40, 50, 60
	α	10, 20, 30, 40, 50, 60, 70, 80, 90
	β	10, 20, 30, 40, 50, 60, 70, 80, 90
	γ	60, 50, 40, 30, 20, 10, 0, -10, -20, -30, -40, -50, -60
	δ	0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
	λ	0.1, 0.01, 0.001
MF-BPR	f	40, 50, 60
	β	0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
	λ	0.1, 0.01, 0.001

Table 4.18: The models' hyper-parameters and their optimal values

Model	Hyper-parameters
iviMMF	$f = 60$, $\lambda = 0.1$, $\alpha = 2$
AMMF(LR)	$f = 60$, $\lambda = 0.001$, $\alpha = 30$, $\beta = 90$, $\gamma = -50$ $\delta = 0.6$
iviMMF	$f = 60$, $\lambda = 0.001$, $\beta = 0.8$

shown in Table 4.17. The optimal values for the model's hyper-parameters are presented in Table 4.18.

Since, on average, about 70% of video views are made from *ivi*'s main page, which is generated using iviMMF, we expect that the evaluation metrics of iviMMF will be higher than those measured on other models. In other words, the data for offline experiments is skewed towards the iviMMF model.

The results of our evaluation are presented in Table 4.19. As it was expected, the iviMMF model showed the best results in terms of offline metrics. The prediction accuracy of AMMF(LR) is slightly worse, while the poorest results are observed on the MF-BPR model. We conjecture that the reason is that MF-BPR was not able to cope with complex dependencies between the selected action types. Taking into account the outcomes of the offline evaluation, we decided to conduct online experiments only with iviMMF and AMMF(LR).

Not all of the available action types can be useful for predicting video views. However, we decided not to identify the optimal subset of action types by using the collected data since they are influenced by iviMMF. Such optimization may not be the best one for the online evaluation. Therefore, for the online experiment, we use the AMMF(LR) model employing information from all the action types.

Table 4.19: Comparison of models in terms of MAP@k and F1@k. Best values are underlined.

Model	MAP			F1		
	@1	@5	@10	@1	@5	@10
iviMMF	<u>0.079</u>	<u>0.065</u>	<u>0.058</u>	<u>0.040</u>	<u>0.061</u>	<u>0.058</u>
AMMF(LR)	0.063	0.053	0.047	0.034	0.053	0.052
MF-BPR	0.031	0.025	0.021	0.016	0.023	0.022

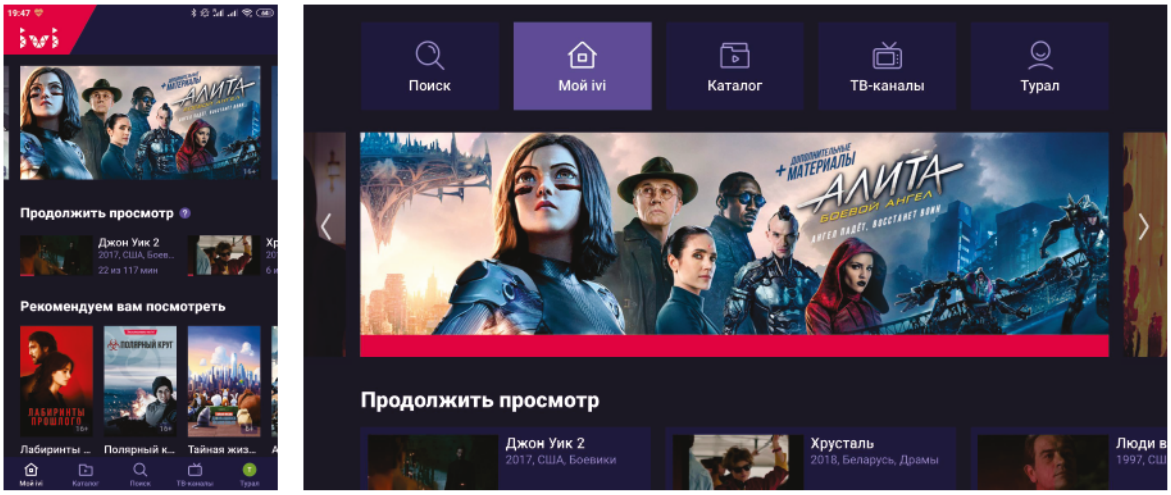

Figure 4.5: The main pages of *ivi* application for smartphones (left) and smartTVs (right).

4.6.4 Results

The main goal of our online evaluation is to show the influence of the prediction models on user behavior. Therefore, during the evaluation, we varied only the prediction component of the RS keeping all other components, such as business rules or the user interface, unchanged.

The behavior of the users varies from platform to platform, where the most significant differences are observed between smartphone and smartTV users. Smartphone users consume video discretely, i.e. they watch videos in short pieces and prefer short content to the long one. On the contrary, smartTV users usually choose content more carefully and watch it to the very end. There is also a difference in the user interfaces for these two platforms (Figure 4.5). For this reason, the evaluation has been conducted on *ivi* subscribers using either smartphone or smartTV applications. The evaluation has been conducted on the subscription users due to the company's business priorities.

For testing, we used the classical AB-test schema. The random subset of users was evenly split into two groups of the same size: test and control. The test and control group users have been exposed to recommendations provided by AMMF(LR) and iviMMF, respectively. The

Table 4.20: The changes in business metrics observed in the test group of the AB-test.

Metric	Smartphone	SmartTV
Conversion to the video view	+0.4%	+0.09%
Conversion to the item purchase	+2.6%*	-0.08%
Conversion to the subscription purchase	+9%*	+7%*
Bounce rate	-6%*	-2%*
Churn rate	-5%*	-4%*

number of users in each group was equal to 30% of all subscribers visited the applications during the test period. The remaining 40% of the subscribers also saw recommendations provided by iviMMF. However, we did not use this user group to calculate the test metrics.

The online experiment was conducted during the 30 days of the summer of 2019. The number of users who entered the AB-test during this period on smartphones and smartTVs was 300K and 440K, respectively. Table 4.20 shows the percentage of changes in business metrics observed in the test group. For example, the conversion to the purchase of an item from recommendations on smartphones increased by 2.6%, and the bounce rate decreased by 6%. Changes marked with an asterisk are statistically significant. To test the statistical significance, a one-sided Z-test with p-value less than 0.001 has been applied.

Throughout the experiment, we collected and analyzed data about what recommendations are shown to the user, how often they change, and how the user interacts with them. The analysis shows that the AMMF(LR) model can automatically recognize and take into account not only positive (e.g., clicks, purchases, views), but also negative implicit feedback (impressions). For example, the model has learned to understand that if the user has seen several times an item in her recommendations, but did not interact with it, then most likely she does not want to watch this item and it should be hidden from the recommendations. In our case, the model believed that the user would not want to watch the movie if she had already seen it in the recommendations more than eight times and never interacted with it. Thus, AMMF(LR) can replace items which the user is not interested in with other relevant and less known to the user items. After seeing such recommendations, users prefer to prolong their subscription, thereby reducing the churn rate.

At the moment, *ivi* is moving towards changing the default recommender to the AMMF(LR) model.

4.7 Conclusions

In this chapter, we presented Automated Multi action type Matrix Factorization (AMMF) that predicts a target user action by leveraging information about actions of multiple types,

temporal delays between actions or the order of actions (predictors). In contrast to Multi action type Matrix Factorization (MMF), AMMF finds correlations between predictors and actions of target type automatically employing machine learning (ML) techniques. The automation makes the process of finding correlations easier and more precise, which in turn increases the prediction accuracy of the model.

AMMF is a hybrid model that consists of two components. The first component is the $\mathcal{M}$ model which predicts the probability that a user will perform a target action on an item given a user-item interaction history. The second component, which is the WR-MF model, employs the first component and the collaboration between multiple users and items to produce the final action predictions. In general, only the $\mathcal{M}$ model is responsible for the incorporation of additional information into AMMF. Although an improvement in the $\mathcal{M}$ model can lead to more accurate predictions, it is not guaranteed due to the limited modelling capacity of the linear factor model. Moreover, the AMMF model captures mainly long-term user preference profiles. For this reason, in the next chapter, we consider a model that has a larger modelling capacity and can take into account both long- and short-term user preferences.

Chapter 5

Personalized Ranking with Long- and Short-term User Preferences

The task of item recommendation is to create a user-specific ranking for a set of items. In the best case, the ranking should match both the users' long-term preferences as well as their more recent interests. For example, consider a user of a career-oriented service (e.g., LinkedIn[1] or XING[2]) who has worked for several years as a software engineer, but is currently interested in the position of a manager. An effective service, which knows the user's employment history, should be able to provide job recommendations satisfying both the user's long-term preferences and her current search interests. Thus, in this example, a position of technical manager in an IT company can be considered as a good recommendation. However, all the multi action type prediction models that we have presented so far do not distinguish between long- and short-term user preferences and therefore can recommend only the postings related to software engineering.

In this chapter, we present a novel multi action prediction model for RSs, which we called LSTP (Long- and Short-Term Preference), that can distinguish between long- and short-term user preferences and use them to predict whether the user will perform a target action on an item. The LSTP model uses the BPR-Opt optimization criterion [84] which directly optimizes the AUC ranking metric [40]. Similarly to Automated Multi action type Matrix Factorization (AMMF), introduced in the previous chapter, LSTP automatically determines correlations between action types and leverages temporal patterns and information about the ordering of actions. Moreover, the model works with user interaction histories which allows to take into account the influence of a user's interaction with one item on the interaction with another. Finally, LSTP can be easily enriched with contextual, user-, and

[1]https://www.linkedin.com/
[2]https://www.xing.com/

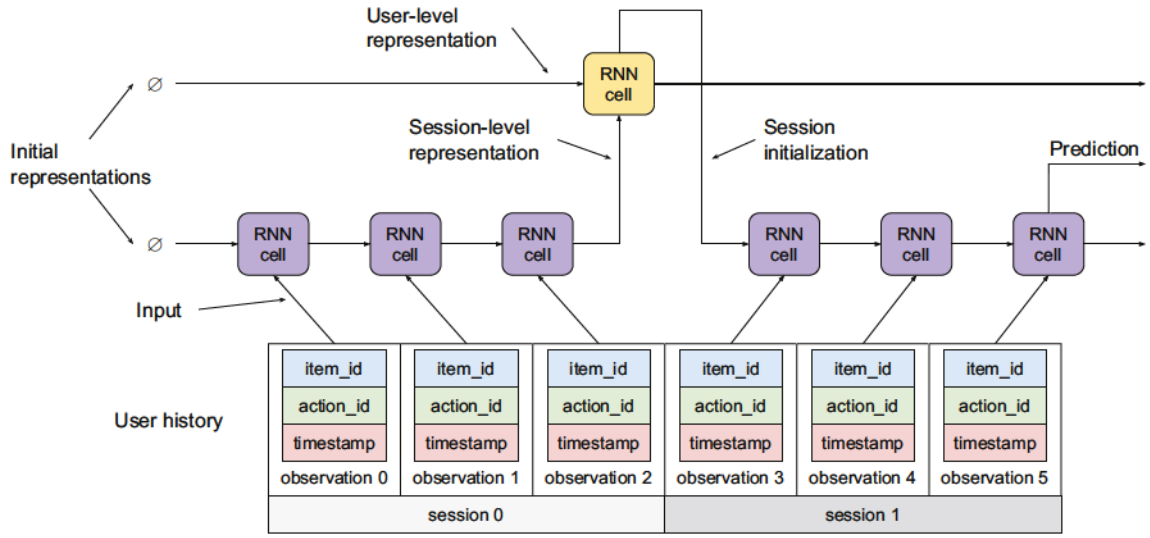

Figure 5.1: Graphical representation of the hierarchical RNN model which consists of the session- and user-level RNNs

item-related information, which can be used to overcome the data sparsity and the cold-start problem [106].

5.1 Action Prediction Based on User Interaction Histories

LSTP predicts whether the user will perform a target action on an item by leveraging user interaction histories. State-of-the-art action prediction models which employ user histories and can capture short- and long-term user preferences use sequence modelling techniques and are usually based on Recurrent Neural Networks (RNNs) [35, 82, 83].

In its basic form, an RNN model inputs a sequence of a user's past actions and outputs predicted scores on items [35]. The higher the score, the more likely the user will perform the target action on the item. As it was mentioned in Section 2.2.2, a potential issue with such an approach is that an RNN needs to compress all the necessary information of an input user interaction history into a fixed-length hidden state. Long histories can complicate the work of the RNN and as a consequence the quality of predictions might deteriorate. This is especially true when the user interaction history consists of actions of multiple types: the more types of actions are used, the longer the history is.

One way to overcome the above-mentioned long histories problem is to employ hierarchical RNNs which model user preferences across sessions [83] (Fig. 5.1). The top level of the hierarchy, which represents the state of the user across sessions, is used to initialize the

bottom level of the hierarchy, which is used to generate recommendations within a session. Thus, the top level models users' long-term preferences, while the bottom level generates recommendation by also taking into account users' short-term interests. Each item a user interacted with is modelled in the RNN with a real-valued embedding vector which has to be learned during the training phase. Given the complexity of the architecture and the desire to avoid overfitting, one needs a sufficiently large amount of data to train the model.

To reduce the amount of data required for the training and to speed up the training process, one can learn item embeddings separately from the final action prediction RNN. In this case, a simpler model which requires less training data can be used to learn the embeddings. The pre-trained embeddings, which can be either fixed (constant) or trainable, are then used in the target RNN. AMMF, presented in the previous chapter, is an example of such a simple model. The advantage of using AMMF is that in comparison to other models building distributed item representations, such as Prod2Vec [27] or Meta-Prod2Vec [103], it learns the item latent feature vectors, which can be considered as embeddings, by leveraging information about actions of multiple types and temporal delays between actions. Moreover, the user vectors learned by AMMF represent the long-term user preferences. They can be used as an analogue of the user-level model in the hierarchical RNNs.

Thus, we propose a novel sequence-aware action prediction model LSTP that employs actions of multiple types and takes into account short- and long-term user preferences to predict whether a user will perform an action of a target type on an item. The core component of LSTP is an RNN which uses f-dimensional user and item factors vectors generated by AMMF to produce the final action prediction. The user vectors, which describe the long-term user preferences, are used to initialize the RNN. The item vectors are used as a part of the whole input to the RNN, which contains the n most recent observations from a user interaction history. Each observation describes an action performed by a user on an item at a certain timestamp. In other words, to predict target actions, LSTP combines the recent user activity with the long-term user preferences by means of an RNN (Fig. 5.2).

Similarly to AMMF, the LSTP model discovers the relationships between actions of different types automatically and exploits temporal patterns and information about the ordering of actions. However, in the case of LSTP, the actions are considered at the level of the user rather than a particular user-item interaction history, i.e. the model can recognize the influence of the user's interaction with one item when predicting the interaction with another.

In general, if the short-term user activity contains only actions of the single type, then one can combine AMMF with simpler sequence-aware techniques, such as Frequent Pattern Mining (FPM) or Markov Chains (MC), into a weighted hybrid model. In this case, for each item, the AMMF and sequence-aware models predict scores which are then combined into

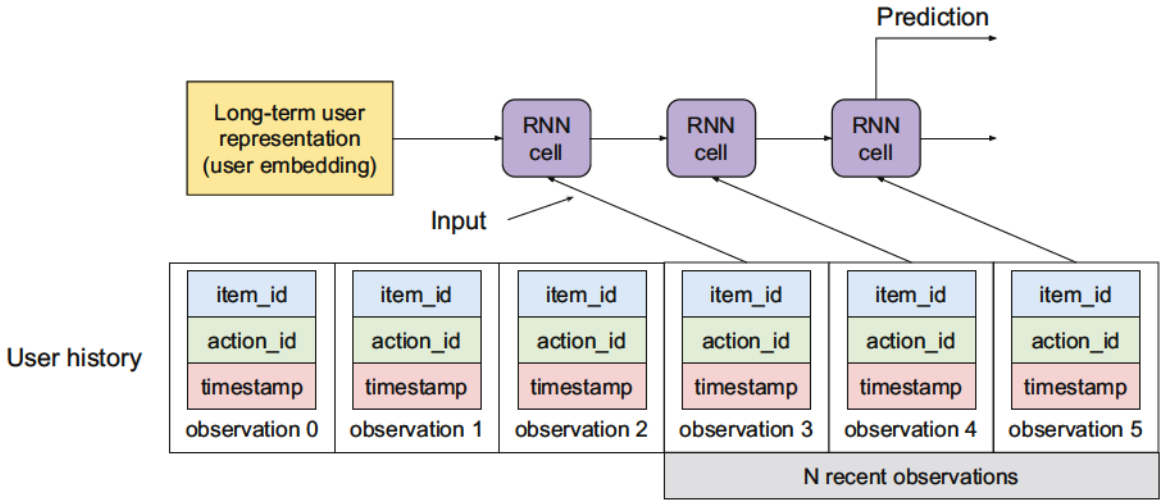

Figure 5.2: Graphical representation of the LSTP model inference. The model combines the recent user activity with the long-term user preferences through an RNN, to predict future user actions.

a single weighted score. The items with the highest weighted score are used to generate the final list of recommendations. However, it is worth to notice that most of the simple sequence-aware techniques cannot easily incorporate contextual information (e.g., time). Moreover, these techniques have usually the limited scalability.

In the next section, we describe in detail the architecture of LSTP, as well as some of its variations. During the evaluation, these variations will help us to understand better how various parts of LSTP affect the prediction quality of the model.

5.2 Action Prediction Model Architecture

To ease the explanation of LSTP's architecture and its variations, we introduce the toy interaction history h_e containing 10 observations of actions performed by the user e on various items (Fig. 5.3). The j^{th} observation of the history $(a_m, i_k, t)^j$ describes an action of type a_m performed on item i_k at timestamp t. Two action types are presented, namely *click* and *reply*. The reply action is the target one. The *four* most recent observations in h_e express the short-term interests of the user e.

During the inference, the LSTP model is initialized by the user embeddings learned by AMMF and employs the most recent observations from a user interaction history as the input. Each observation contains information about an action performed by a user on an item at a certain timestamp. Depending on whether the information about the user, the type

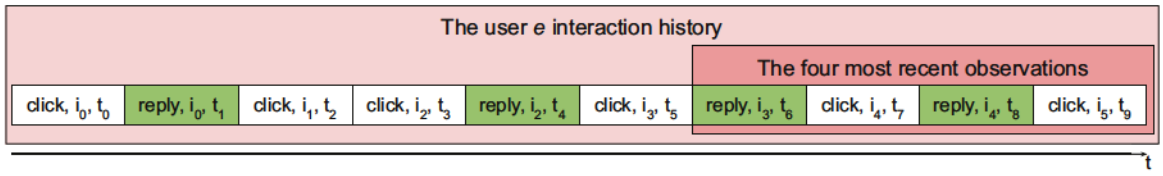

Figure 5.3: The toy interaction history h_e which contains observations of clicks and replies performed by the user e on various items

Table 5.1: The features of the LSTP model and its variations.

	Uses actions of multiple types	Distinguishes action types	Takes timestamps into account	Initializes the RNN with user embeddings
LSTP	+	+	+	+
aLSTP	+		+	+
tLSTP	+	+		+
iLSTP	+			+
rLSTP			+	+
STP	+	+	+	

of action, or the time is taken into account, one can consider different modifications of the LSTP model. Table 5.1 summarizes the differences of the models which we describe in detail in the remained of this section.

5.2.1 Predicting Target Actions Using User and Action Embeddings

LSTP is a neural network model that consists of the input, recurrent, feedforward, and output layers (Fig. 5.4). The input to LSTP is a user identifier and the most recent observations from the user's interaction history. For a given user identifier, the model looks up for the corresponding user embedding and use it to initialize the recurrent layer closest to the input layer. Other recurrent layers, if there are any, are initialized with zero states.

Each observation contains an action type identifier, an item identifier, and the time when the action has been observed. Similarly to users and items, each action type is modelled with a separate embedding. Since there are usually only a few action types in a dataset, instead of pretraining action type embeddings one can learn them during the RNN's training step. The input layer is the concatenation of item and action embeddings and the timestamp. The recurrent layers are Gated Recurrent Units (GRUs) [13]. Finally, the output layer contains the predicted ranking scores of the items. The higher the score, the more likely the user will perform the target action on the item. In its basic configuration, besides the input and output layers, the LSTP model has only one recurrent layer.

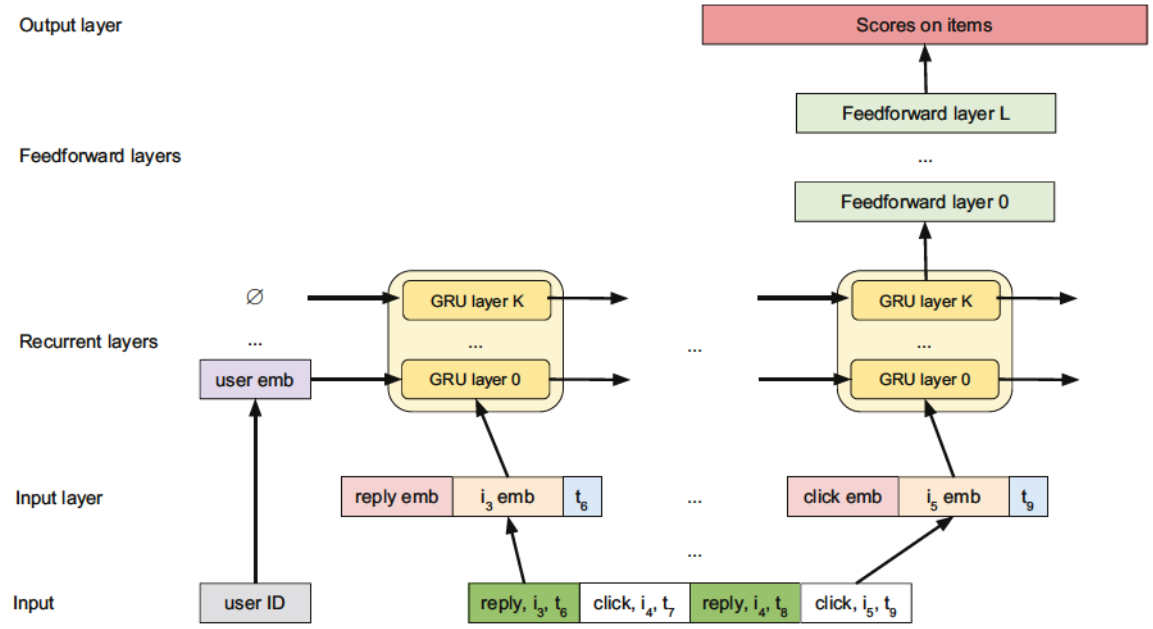

Figure 5.4: The LSTP model architecture

The proposed architecture has a number of advantages. The LSTP model incorporates additional information available in various types of user actions through the user, item, and action embeddings. This does not require knowledge of the nature of action types, hence the model is domain-independent. The recurrent layers take into account the order of the user actions and consider them on the level of a user rather than a user-item interaction history, i.e. the model can recognize the influence of a user's interaction with one item on interaction with another. Moreover, the recurrent layers exploit the long-term user preferences and short-term user interests. Finally, this architecture allows taking into account the time when an action was performed without the need to discretize the timestamps.

It is worth noting that the action type and timestamp can be considered as contextual information for the observation of the item [16, 93].

5.2.2 Ignoring Contextual Information While Predicting Target Actions

In the LSTP model, we conjecture that action embeddings and timestamps can help us to incorporate additional information and to improve the target action prediction accuracy. However, some models from the research literature ignore this type of information; they do not distinguish the action types [83] or consider only the ordering of the actions by neglecting the timestamp values [35].

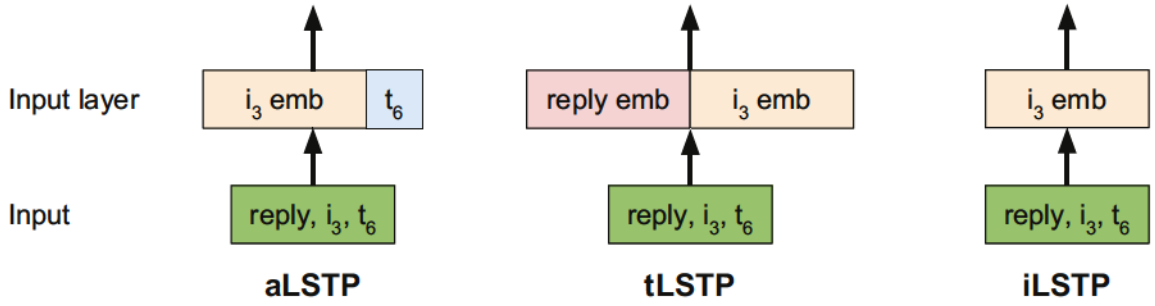

Figure 5.5: Three modifications of the LSTP architecture (only the input part): aLSTP (left), tLSTP (center), iLSTP (right)

To check the usefulness of different contextual information, we propose three modifications of LSTP, namely aLSTP, tLSTP, and iLSTP (Fig. 5.5). The aLSTP model considers observations of actions of multiple types but does not distinguish the action types. Hence, its architecture does not include the action embeddings. The tLSTP model distinguishes action types, but it ignores the time information. Finally, the iLSTP model neither distinguishes the action types nor uses information about the time.

All the models presented above employ user and item embeddings generated by AMMF which learns the embeddings by leveraging information about actions of multiple types and temporal delays between them. Therefore, to make the comparison of the models fair and to localize it to the level of the RNN input, we used user and item embeddings generated by the same AMMF model. We recall that the evaluation of the dependence of AMMF on types of actions and time is presented in Section 4.5.

5.2.3 Predicting Target Actions Using Only Recent User Actions

In this section, we introduce another modification of the LSTP model, which we refer to as STP, that predicts whether the user will perform a target action on an item by leveraging only short-term user activity. Instead of employing user embeddings, the STP model initializes all the RNN layers with zero states.

Except for the difference in the input layer, the STP model is similar to GRU4Rec proposed by Hidasi et al. [35]. GRU4Rec uses the one-hot item encodings as input to the recurrent layer, while the recurrent layer in STP inputs concatenations of item and action embeddings and timestamps. We use this model to understand better how the most recent user actions influence the prediction quality of LSTP.

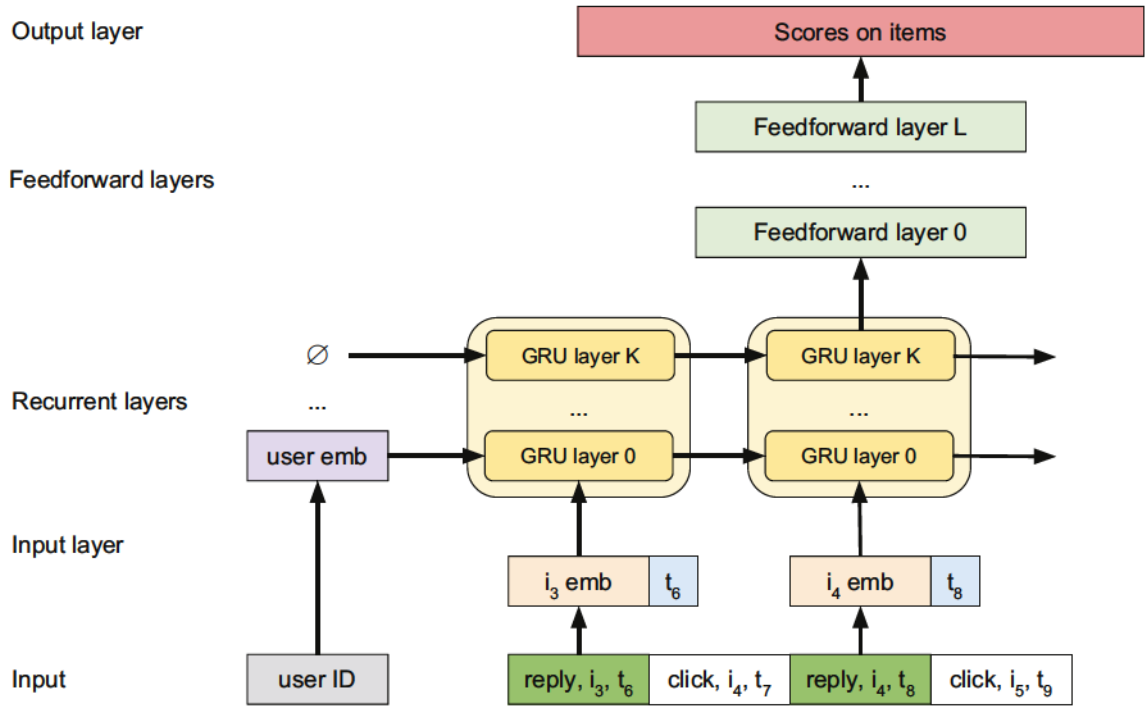

Figure 5.6: The rLSTP model which takes into account only actions of the target type

5.2.4 Predicting Target Actions Using Only Actions of the Target Type

Finally, we consider the variation of LSTP which takes into account only actions of the target type. To train the model, which we call rLSTP, we keep only target action observations in user interaction histories. Thus, since the histories now contain actions of a single type, the rLSTP model does not need action embeddings. Moreover, the user and item embeddings for rLSTP are generated by the WR-MF model, which is also trained by using only the target action data. Thus, we guarantee that rLSTP does not know anything about user actions of non-target types. The model is designed to show how much the prediction result can deteriorate if one ignores information about other types of actions.

To make rLSTP comparable with LSTP, for a given user interaction history, we first trim the n most recent user actions of all the types and then use only the observations of the target actions as the input to rLSTP. Thus, the models operate with information from the same period.

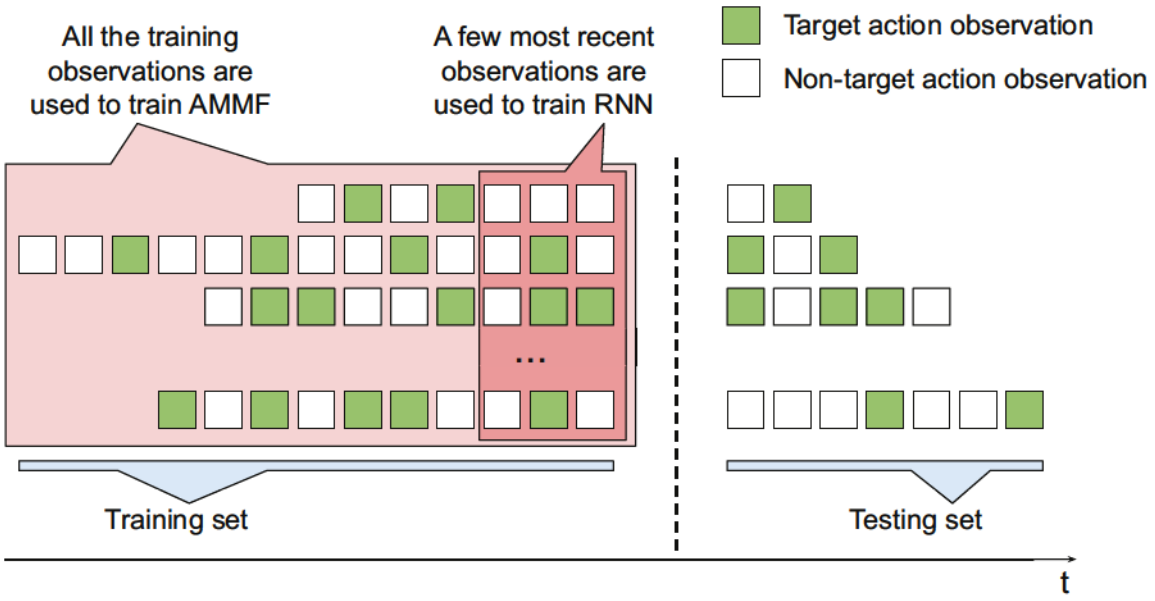

Figure 5.7: Data used for the training of the LSTP model and its variations

5.3 Training Procedure

The training procedure for all the proposed models consists of two steps. At the first step, we use a MF-based model to learn the user and item latent feature vectors from the user-item interactions observed in the training set. In the case of rLSTP, the vectors are learned by the WR-MF model presented in Section 4.4.1. For all the other models, we use the vectors generated by the AMMF(LR) model presented in Section 4.4.3. The vectors learning procedures for WR-MF and AMMF(LR) are described in the corresponding sections. At the second step, we use vectors, along with the last n observations from user interaction histories available in the training set, to train the RNN component (Fig. 5.7). The component is optimized for the BPR-Opt criterion [84].

Traditionally, in RSs, RNNs are trained by applying the sequence to sequence (seq-to-seq) approach [35, 93]. The approach assumes that an RNN is trained to predict for a given input sequence the target sequence, which is the same as the input sequence, except it is shifted by one time step into the future. For example, given a sequence of items $s = [i_0, i_1, \ldots, i_n]$, the RNN is trained to predict from the input sequence $[i_0, i_1, \ldots, i_{n-1}]$ the output sequence $[i_1, \ldots, i_n]$ (Fig. 5.8a). Thus, at each time step, the model learns to predict the item which expected to be observed at the next step.

The seq-to-seq approach can also be applied to sequences consisting of user actions of different types [93]. As before, at each time step, the model learns to predict the item expected

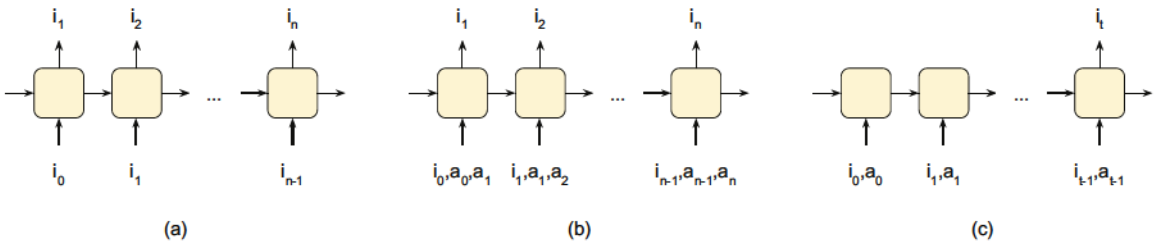

Figure 5.8: Different approaches for an RNN training: (a) seq-to-seq, (b) seq-to-seq conditioned on action types, (c) seq-to-vec, where i_t denotes the item on which the target action type was performed

at the next step. However, in this case, the prediction is conditioned by the expected action type. For instance, for a given sequence $s = [(i_0,a_0),(i_1,a_1),...,(i_n,a_n)]$, the RNN learns to predict from the input sequence of observations $[(i_0,a_0),(i_1,a_1),...,(i_{n-1},a_{n-1})]$ the output sequence of items $[i_1,...,i_n]$ for the corresponding types of actions $[a_1,...,a_n]$ (Fig. 5.8b). During the inference, in addition to the input sequence, the target action type is passed to the RNN. This additional information helps the model to output items on which this type of action is more likely to be performed.

Unfortunately, the direct use of the seq-to-seq approach for multi action prediction models may not always be preferred. This is due to the fact that user actions of different types are observed in unequal proportions in the training set. As a result, the model can be well trained to predict items only for the prevailing type of action, which is often not the target one (e.g., clicks are observed more often than purchases). Hence, the prediction quality for the actions of the target type may be poor. To avoid this problem, we decided to train the RNN component of LSTP using the sequence to vector (seq-to-vec) approach. The seq-to-vec approach is a special case of seq-to-seq where the model is trained to predict for an input sequence only one next item. In our case, the predicted item is the one on which the target action was observed. Therefore, the LSTP model learns to find items on which only the target action is expected to be performed (Fig. 5.8c). Though this approach solves the problem of the action types imbalance, when the target type of action is predominant in the training set, it may converge more slowly and lead to less optimal results. This is because in our seq-to-vec approach, the model will be updated less frequently and with less force, which in turn can affect the quality of its prediction.

Let us denote with H the training set of user interaction histories. We train the RNN to predict whether a user will perform a target action on an item by considering only the n most recent user actions. Therefore, for each training user interaction history $h_u \in H$ we keep

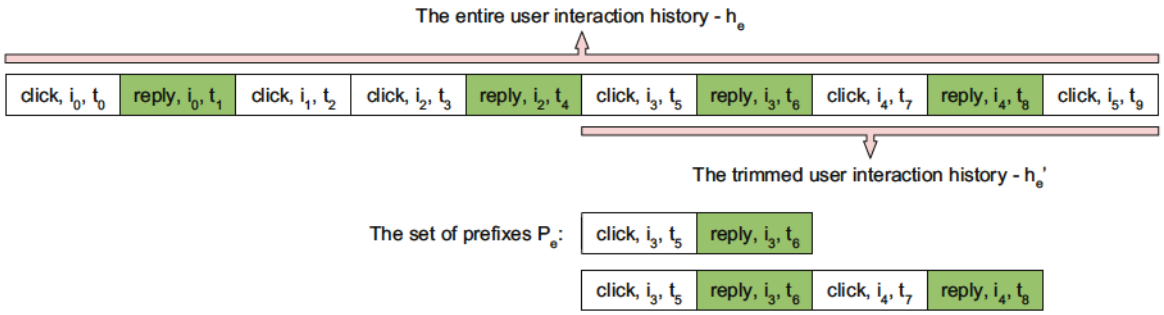

Figure 5.9: Generating a set of training prefixes from a user interaction history

at most the $n+1$ last observations (the length of some histories can be less than $n+1$). We define as h'_u the trimmed version of h_u, while H' is the set of trimmed histories that contain at least one action of the target type. To force the RNN to generalize better, we split each training history $h'_u \in H'$ into a set of prefixes P_u where each prefix p_u ends with an observation of the target action. The RNN is trained to predict the last (target action) observation in p_u given all the previous observations.

For example, let us consider the toy user history h_e presented in the previous section. Assuming that $n = 4$, the history h_e can be split into the set prefixes P_e as it is shown in Fig. 5.9. Using the first prefix, the model is trained to predict that the user e will perform a reply action on i_3 given only the previously observed click on i_3 at time t_5. Accordingly, the second prefix is used to train the model to predict that the user e will perform a reply action on i_4 given the three previous observations.

As we mentioned before, the RNN component is optimized directly for ranking using the pairwise BPR-Opt criterion. The criterion assumes that each training item i, which the user u replied to, is paired with the item j that has not been replied by u yet. The item i is considered as a positive example, while j as negative for the user u. Using the optimization criterion, the model learns to rank higher the positive examples.

As noted in Section 2.2.1, there are other ranking optimization criteria which can be applied for LSTP. Hidasi et al. [34] showed that the choice of the loss function has a significant effect on the performance of an RNN and the quality of its prediction. We intend to explore the optimal optimization criterion for LSTP in our future work.

To train the LSTP model and its variations, we use a stochastic mini-batch version of the gradient-based Adam algorithm [48]. During the training process, the item and user embeddings are fixed. A mini-batch for all the models except STP contains m records, where each record contains a user identifier, a set of recent user action observations, a positive item identifier, and a sample of negative item identifiers of size l.

5.4 Evaluation

This section presents the empirical evaluation of the proposed LSTP model and its variations. Two evaluation scenarios are considered. In the first scenario, the variations of LSTP are compared with each other on the XING dataset (Section 4.3.1). The purpose of this scenario is to check the usefulness of different contextual information for the action prediction in this target dataset. In the second scenario, the LSTP model is compared with the Concat-Mult-GRU-RNN (CRNN) model proposed by Smirnova et al. [93] on the three datasets introduced in Section 4.3. CRNN is a contextual multi action RNN model which is trained using the seq-to-seq approach. The prediction of the model is conditioned by the action type which is expected to be observed on an item (i.e., the target action type) and the time context when the action is expected to happen. To make the LSTP and CRNN models comparable, during the evaluation, the expected time context for the CRNN is set to the earliest timestamp observed in the testing set. In this case, both LSTP and CRNN do not know anything about the testing set. The quality of the models is assessed by measuring Mean Average Precision at k (MAP@k) and F1-score at k (F1@k) which are explained in detail in Section 4.5.2. Analogously to the previous chapters, in our evaluation experiments $k \in \{1,3,5,7,10\}$. We use the same data splitting protocol as described in Section 4.5.1.

5.4.1 Comparing LSTP Variations

In this section, we provide the results of the offline experiments conducted on the XING dataset in which compare the variations of the LSTP model, namely LSTP, aLSTP, tLSTP, iLSTP, STP, and rLSTP, with each other. During the evaluation, for each user in the testing set a model generates a list of items (recommendations) which are predicted to be job postings to which the user replied. The postings are then sorted by the predicted score and the top-k items are selected.

To avoid overfitting and to speed up the training process, all the proposed models employ pre-trained user and item embeddings, which are the user and item latent feature vectors generated either by WR-MF for rLSTP or by AMMF(LR) for the other variations. The WR-MF and AMMF(LR) models are trained to predict whether a user will perform an action of the target type on an item. To train the models, we use the same procedure to fine-tune the hyper-parameters as it is presented in Section 4.5. This allows us to compare the MF-based models presented in the previous chapters with LSTP and its variations. We report the optimal values for the WR-MF and AMMF(LR) models' hyper-parameters for the target action prediction task in Table 5.2.

Table 5.2: The MF-based models' hyper-parameters and their values

Model	Hyper-parameters
WR-MF	$f = 100, \lambda = 0.1, \alpha = 100$
AMMF(LR)	$f = 160, \lambda = 0.1, \alpha = 200, \beta = 160, \delta = 0.8$

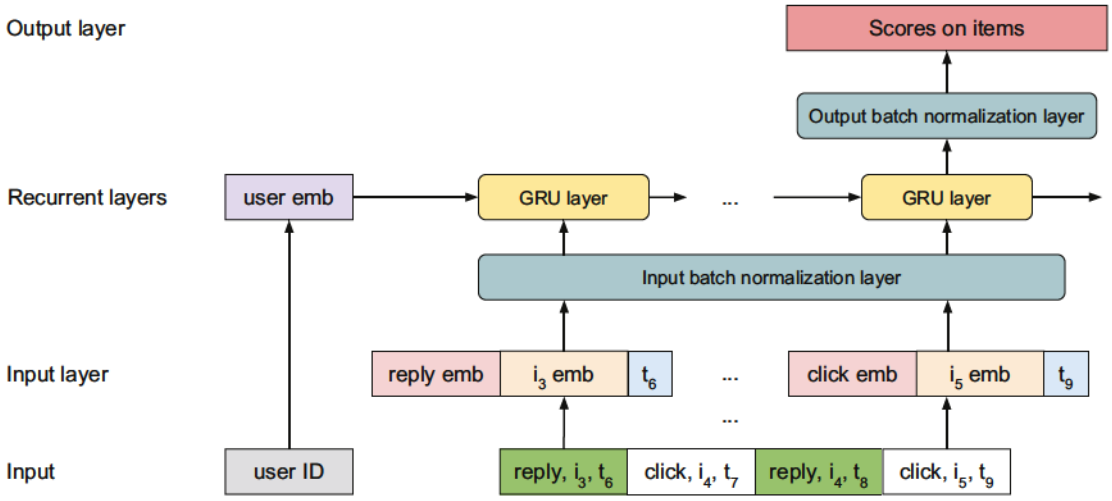

Figure 5.10: The optimal architecture of the LSTP model for the evaluation on the XING dataset

In our experiments, all the models have almost the same optimal architecture and hyper-parameters which have been tuned by optimizing the prediction accuracy of the models on the validation set in terms of F1@5. For all the models, we use a neural network architecture which contains one recurrent layer and does not contain feedforward layers (Fig. 5.10). To improve the performance and stability of the network, we employ batch normalization layers [39] between input and recurrent, and recurrent and output layers.

The models are optimized for the BPR-Opt criterion using the stochastic mini-batch gradient descent. The grid search [25] method for parameter optimization has been applied. Parameter values on which the search was carried out are shown in Table 5.3. Here *n_units* is the number of recurrent units, γ is the learning rate, f_a is the size of action embeddings, λ_a is the regularization parameter for the action embeddings, and l the negative sample size for the BPR-Opt criterion. The optimal values for models' hyper-parameters are presented in Table 5.4, where *n_iter* is the optimal number of training iterations. The optimal length of the recent history analyzed by the models is $n = 5$. For all the models, the training batch size is equal to 256.

Table 5.3: Parameter values for the grid search optimization of the LSTP variations

Hyper-parameter	Values
n_units	32, 64, 80, 100, 128, 160, 192
γ	0.1, 0.01, 0.001, 0.0001
f_a	2, 4
λ_a	0.1, 0.01, 0.001
l	50, 100, 200

Table 5.4: The optimal values for the LSTP variations' hyper-parameters

	LSTP	aLSTP	tLSTP	iLSTP	STP	rLSTP
n_units	100	128	80	100	160	100
γ	0.0001	0.0001	0.0001	0.0001	0.001	0.0001
f_a	2	-	2	-	2	-
λ_a	0.1	-	0.1	-	0.1	-
l	100	100	100	100	100	100
n_iter	40	30	35	30	20	25

Since the neural network weights are initialized randomly, it may lead to the situation when the order and content of the recommendations provided by the models vary from training to training. That is why the final score for each metric is calculated as an average over the five consecutive model evaluations. The standard deviation of scores is on average within the $[0.001, 0.002]$ interval. The highest standard deviation is usually observed for $k \in \{1,3\}$ and reaches its maximum 0.0036 in the case of iLSTP model for F1@1.

The evaluation results are reported in Table 5.5. To simplify the comparison, we split the table into two parts and added also the performance of the WR-MF and AMMF(LR) models. The upper part of the table presents the models that predict replies by leveraging information only about reply actions, while the lower part presents the models that use observations of actions of multiple types.

Just like multi action AMMF(LR) significantly outperforms single action WR-MF, the multi action LSTP model significantly outperforms single action rLSTP. This shows once again that the incorporation of multiple action types into the action prediction models can be beneficial. At the same time, the rLSTP model which extends WR-MF by explicitly considering short-term user activities performs better than the stand-alone WR-MF which combines long- and short-term user preferences in its own way. Similarly, the LSTP model outperforms the stand-alone AMMF(LR). Thus, by taking into account the short-term user activity, one can improve the prediction quality of the model even more. On the other hand, it might not be enough to use only short-term user interests to provide good recommendations.

Table 5.5: Comparison of models in terms of MAP@k and F1@k. Best values are underlined. Values in bold highlight the cases when the performance difference between LSTP and the next best model is statistically significant with $p < 0.05$ (one-sided t-test).

Model	MAP					F1				
	@1	@3	@5	@7	@10	@1	@3	@5	@7	@10
WR-MF	0.022	0.039	0.045	0.047	0.049	0.026	0.040	0.039	0.035	0.031
rLSTP	0.037	0.056	0.061	0.063	0.065	0.045	0.050	0.044	0.040	0.034
AMMF(LR)	0.027	0.043	0.052	0.056	0.060	0.030	0.040	0.047	0.045	0.040
STP	0.015	0.023	0.025	0.025	0.027	0.019	0.023	0.019	0.016	0.014
iLSTP	0.041	0.063	0.068	0.070	0.073	0.050	0.056	0.049	0.043	0.037
tLSTP	0.041	0.064	0.069	0.072	0.074	0.049	0.058	0.051	0.045	0.038
aLSTP	0.043	0.065	0.070	0.072	0.074	0.052	0.058	0.050	0.043	0.037
LSTP	**0.045**	0.066	**0.072**	**0.075**	**0.077**	**0.054**	0.059	0.051	0.045	0.040

In our experiments, the STP model performs the worst. We conjecture that this is because a recurrent layer, which has limited expressiveness, cannot capture well the dependencies between different actions and items when item embeddings are fixed, and input is short and may contain repeated observations of the same item. To validate this hypothesis, we propose two modifications of STP which we refer to as STPt5 and STPt15. In both models the item embeddings are trainable. The STPt5 model provides recommendations by considering only the five most recent user actions, while STPt15 considers the 15 most recent action observations. The result presented in Table 5.6 shows that though the quality of the STP model can be improved by making item embedding trainable, it is still necessary to provide the model with information about the long-term user activity.

Table 5.6: The comparison of the STP models with fixed and trainable item embeddings

Model	MAP					F1				
	@1	@3	@5	@7	@10	@1	@3	@5	@7	@10
AMMF(LR)	0.027	0.043	0.052	0.056	0.060	0.030	0.040	0.047	0.045	0.040
STP	0.015	0.023	0.025	0.025	0.027	0.019	0.023	0.019	0.016	0.014
STPt5	0.032	0.046	0.048	0.049	0.049	0.039	0.040	0.031	0.025	0.019
STPt15	0.033	0.050	0.053	0.055	0.057	0.040	0.045	0.038	0.033	0.028
LSTP	0.045	0.066	0.072	0.075	0.077	0.054	0.059	0.051	0.045	0.040

Among the models leveraging actions of multiple types, the LSTP model performs the best. In general, the more information is available to the model, the better it performs. However, the performance difference between LSTP and the next best model is statistically significant with $p < 0.05$ (one-sided t-test) only for MAP@$\{1,5,7,10\}$ and F1@1. Based on the evaluation results, it can be concluded that the use of RNN, which takes into account

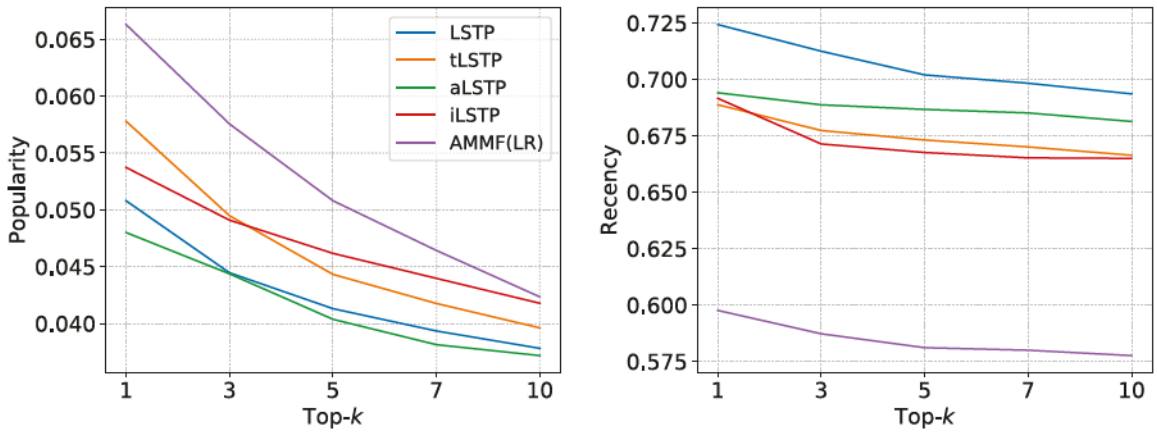

Figure 5.11: The popularity and recency of the top-k recommendations provided by the models

the short-term user activity, together with the BPR-Opt criterion can improve the quality of ranking and thus increase the value of MAP and F1@k for smaller values of k. At the same time, the use of pre-trained latent factors vectors allows to take into account the more long-term user preferences, affecting F1@k for larger values of k.

To show the features of the models, we plotted the statistics of the top-k items recommended by the models (Fig. 5.11). In particular, we used two metrics, namely *popularity* and *recency*. We define the popularity of an item as the number of times it has been observed in the training set. The recency of an item is a value that is inversely proportional to the average amount of time passed since the interactions with this item. Both metrics are normalized to the $[0,1]$ interval. The closer the popularity to 0, the less popular is the recommended item. Analogously, the closer the recency to 1, the more recently users have acted on the item. Typically, items are considered as good recommendations for a user if they match the user's preferences and have low popularity and high recency. Such items can lead to new discoveries. In contrast, with popular items, there is a high probability that the user already interacted with them and might be not interested in them. While items that the user interacted with some time ago but never completed the consumption can be reminded with more straightforward approaches similar to those proposed by Lerche et al. [56].

Fig. 5.11 shows that in general the LSTP model and its variations recommend more recent and less popular items than the AMMF(LR) model. The LSTP and aLSTP models, which employ information about timestamps, recommend the least popular and the most recent items. The difference between LSTP and tLSTP in terms of popularity and recency is statistically significant with $p < 0.005$ (two-sided t-test). At the same time, the use of

Table 5.7: Parameter values for the grid search optimization of the CRNN model

Hyper-parameter	Values
f	20, 40, 60, 80, 100
n_units	20, 40, 60, 80, 100
n_layers	1, 2, 3
γ	0.1, 0.01, 0.001, 0.0001

Table 5.8: The RNN-based models' hyper-parameters and their optimal values for the XING, Rekko, and Taobao datasets

Model	Dataset	Hyper-parameters
LSTP	XING	$n\_units = 100$, $f_a = 2$, $\lambda_a = 0.1$, $l = 100$, $n = 5$, $n\_iter = 40$, $\gamma = 0.0001$
	Rekko	$n\_units = 120$, $f_a = 2$, $\lambda_a = 0.1$, $l = 300$, $n = 5$, $n\_iter = 40$, $\gamma = 0.0001$
	Taobao	$n\_units = 160$, $f_a = 2$, $\lambda_a = 0.1$, $l = 100$, $n = 5$, $n\_iter = 30$, $\gamma = 0.001$
CRNN	XING	$f = 40$, $n\_units = 40$, $n\_layers = 1$, $n = 25$, $n\_iter = 30$, $\gamma = 0.01$
	Rekko	$f = 40$, $n\_units = 60$, $n\_layers = 1$, $n\_iter = 10$, $n = 25$, $\gamma = 0.1$
	Taobao	$f = 60$, $n\_units = 60$, $n\_layers = 1$, $n\_iter = 15$, $n = 25$, $\gamma = 0.05$

action embeddings increases the prediction quality of the RNN-based models but can slightly increase the popularity of the recommended items.

5.4.2 Comparing RNN-based Multi Action Prediction Models

In the second evaluation scenario, we compare LSTP with Concat-Mult-GRU-RNN (CRNN) [93] using all the datasets presented in Section 4.3. Analogously to the previous sections, the models' prediction accuracy has been assessed in terms of MAP@k and F1@k where $k \in \{1,3,5,7,10\}$.

The grid search [25] method was applied for parameter optimization. The sets of the LSTP and CRNN models' hyper-parameter values for the search are shown in Table 5.3 and Table 5.7 respectively. These sets are the same for all the datasets. In the case of LSTP, the AMMF(LR) model is optimized using the procedure presented in Section 4.5. In the case of CRNN, there are four parameters to optimize: the item embedding size f, the number of recurrent units n_units, the number of recurrent layers n_layers, and the learning rate γ. The optimal values for LSTP's and CRNN's hyper-parameters are presented in Table 5.8. Parameter n_iter corresponds to the optimal number of training iterations, while n refers to number of observations input to the RNNs. For the CRNN model, the optimal value for n, which we identified during the cross-validations step, is 25. This value correlates with the average number of interactions for a user in the datasets (Table 4.16).

Table 5.9: The comparison of LSTP and CRNN in terms of MAP@k and F1@k on the XING, Rekko, and Taobao datasets. Best values are underlined.

		XING		Rekko		Taobao	
		LSTP	CRNN	LSTP	CRNN	LSTP	CRNN
MAP	@1	0.045	0.038	0.021	0.029	0.033	0.033
	@3	0.066	0.050	0.033	0.042	0.052	0.040
	@5	0.072	0.056	0.038	0.048	0.060	0.042
	@7	0.075	0.058	0.041	0.051	0.063	0.043
	@10	0.077	0.059	0.043	0.054	0.066	0.044
F1	@1	0.054	0.046	0.032	0.046	0.036	0.035
	@3	0.059	0.044	0.054	0.064	0.044	0.028
	@5	0.051	0.040	0.061	0.068	0.042	0.022
	@7	0.045	0.034	0.063	0.068	0.038	0.018
	@10	0.040	0.029	0.062	0.066	0.033	0.014

Table 5.9 provides the results of the comparison of LSTP and CRNN in terms of MAP@k and F1@k on the three considered datasets. The LSTP model outperforms CRNN on the XING and Taobao datasets, while CRNN performs better on the Rekko dataset. In order to shed light on the results, we would like to refer once again to the statistical analysis presented in Table 4.16. As we mentioned before, in contrast to the LSTP model, CRNN is trained using seq-to-seq approach which can be more suitable for the cases when the target action type prevails in the training set. This explains why CRNN is superior to LSTP on the Rekko dataset, where target actions account for 61% of all the observations. At the same time, CRNN performs the worst on the Taobao dataset in which the proportion of the target actions is the smallest and equal to only 15%.

Thus, the results of this section show that among other advantages, the LSTP model can provide accurate predictions in the cases when actions of the target type are not dominant in a dataset. However, if the actions of the target type prevail, we conjecture that training the RNN component of LSTP using the conditional seq-to-seq approach can improve the quality of the results. We plan to check this hypothesis in our future works.

5.5 Conclusions

In this chapter, we have presented a novel sequence-aware action prediction model that predicts a target user action by leveraging information about the actions of multiple types. The LSTP (Long- and Short-Term Preference) model discovers the relationships between actions types automatically and exploits the temporal patterns and information about the ordering of actions. Unlike the models presented in Chapters 3 and 4, LSTP can distinguish

between long-term and short-term user preferences and use them to predict whether a user will perform a target action on an item.

Since the task of item recommendation is to create a user-specific ranking for a set of items, the LSTP model directly optimizes the AUC ranking metric [40]. We assume that in addition to considering short-term user preferences, this can be yet another reason why LSTP performs better in terms of MAP@k and F1@k than AMMF(LR), which is optimized for regression. The conducted evaluations showed that in comparison to other models presented in the previous chapters the LSTP model can recommend relevant and, at the same time, novel items.

The core component of LSTP is an RNN which employs user and item factors vectors generated by Automated Multi Action Type Matrix Factorization (AMMF). The user vectors, which describe the long-term user preferences, are used to initialize the RNN. The inputs to the RNN are the n most recent actions performed by a user on items, where each action observation is described by an item vector and additional contextual information (e.g., the action type and time). Thus, to predict target actions, LSTP combines the recent user activity with the long-term user preferences using the RNN. Such an approach allows to reduce the amount of data required for the training and to speed up the training process. In general, both user and item vectors can be easily enriched with additional contextual, user-, and item-related information.

The LSTP model is a domain-independent multi action prediction model for RSs. It can automatically determine correlations between action types and leverage temporal patterns and information about the ordering of actions. At the same time, due to the complexity of the model, it can be difficult to interpret the predicted results. In our future works, we plan to apply the attention mechanism [62] to the RNN component which will help us to weight the input information and to understand better which user actions influenced the prediction most of all.

Chapter 6

Conclusions

To provide personalized item suggestions, users' preferences have to be known by the Recommender System (RS). These preferences are inferred from feedback provided by users on items which can be either *explicit* or *implicit*. Unlike explicit feedback, where a user provides her preferences in the form of ratings or explicit likes, when implicit feedback is used, the system must infer the user's preferences from the user's action (clicks, purchases, views, etc.).

For a long time, research on RSs was mainly focusing on models utilizing explicit feedback. However, in many real-world scenarios obtaining preference information explicitly can be either difficult or even impossible. For this reason, various approaches employing implicit feedback datasets for RSs have emerged in recent years. These approaches assume the existence of user behavioral patterns that highly correlate with explicit user feedback and ultimately with the user preferences. In particular, they implicitly assume that if a user performs a *target action* on an item, then she is interested in this item. Therefore, implicit feedback models predict on which items the user will perform a target action. These items are then considered as good recommendations.

To make predictions, implicit feedback models examine users' actions. Generally, in the context of user profiling, researchers rely on one type of specific user interactions like item views or purchases. Single action prediction models, while providing reliable results, ignore information that could be extracted from other types of actions, for instance, clicks or bookmarks. This information can be used to build more comprehensive user models that describe more precise users' preferences and, as a consequence, allow creating better RSs.

The use of information about actions of multiple types in RSs requires addressing challenges related to the identification of correlations between action types, and the identification of predictive actions for a target action type. In this thesis, we explored some of these challenges that have not been addressed in previous research. In particular, we focused

on the tasks of building domain-independent multi action prediction models for RSs that can automatically elicit predictive action types for a given target action type, determine correlations between action types, and leverage temporal patterns and information about the ordering of actions.

6.1 Summary

In the introductory chapters of this thesis, we provided an overview of the historical categorization of implicit feedback types and the current state of research on single and multi action prediction models for RSs. We showed that although some models incorporating actions of multiple types have been recently proposed in the research literature, most of them are based on domain-dependent heuristics which may overlook subtle relations between actions of different types. Moreover, these models capture only long-term user preference profiles and do not take into account temporal relations between actions, such as the order of the actions and the time delays between them.

In Chapter 3, we presented our general prediction model (MMF - Multiple action types Matrix Factorization) which is designed to predict a target user action by leveraging information about actions of multiple types. Even though the model exploits correlations between different action types through heuristics determined by exploratory analysis of the data, we used it as the basis for the more advanced approaches presented later. For example, an extension of MMF called Automated Multi action type Matrix Factorization (AMMF), which we presented in Chapter 4, finds correlations between different action types automatically using machine learning (ML) techniques. The automation made the process of finding correlations between action types easier and more precise, which in turn increased the prediction accuracy of the model. Moreover, it allowed to take into account additional valuable information, such as temporal delays between actions or the order of actions.

To provide relevant recommendations, RSs should take into account not only long-term user preferences but also their more recent interests. However, neither MMF nor AMMF distinguishes between these two types of information. For this reason, in Chapter 5, we presented a novel sequence-aware action prediction model, which we named LSTP (Long- and Short-Term Preference), that takes into account long-term user preferences as well as short-term user interests to predict whether a user will perform a target action on an item. The LSTP model is a domain-independent multi action prediction model for RSs which can automatically determine correlations between action types and leverage temporal patterns and information about the ordering of actions. This model is the culmination of this thesis since it covers all the challenges we have been focused on.

As we mentioned in the Introduction, in this thesis we addressed the following research questions:

RQ1: How to generate relevant recommendations by exploiting information about the full variety of actions that a user may perform while interacting with the system?

RQ2: How to discover automatically dependency relations between action types?

RQ3: How to exploit the temporal patterns and information about the ordering of the users' performed actions?

Taking these questions into consideration, in Chapters 3 to 5, we have presented various innovative techniques which can be used to generate relevant recommendations by leveraging information about user actions of multiple types (**RQ1**). Then, in Chapters 4 to 5, we introduced multi action prediction models that can automatically determine correlations between action types using ML techniques (**RQ2**). Moreover, to predict whether a user will perform a target action on an item, the LSTP model was presented in Chapter 5. The model exploits the temporal patterns and ordering of actions and takes into account short-term user interests (**RQ3**).

All the conducted studies presented in this thesis have shown a significant improvement in the target action prediction accuracy when multi-type implicit feedback is used. They also showed that it is possible to create more effective RSs that will not require users to specify their preferences explicitly and will use extensive and presumably information-rich data generated by user activity. Such systems can be then utilized in areas where users do not or cannot provide only explicit feedback to the system (e.g., news portals, streaming music and video services, tourism applications).

6.2 Future Directions

Over the past three decades, a variety of action prediction models have been developed for implicit feedback RSs. In pursuit of better predictions, each subsequent model became more and more complicated and less domain-independent and interpretable. This led to a situation where the behavior of models ceases to be intuitive, while the evaluation and comparison of models become uninformative. In this thesis, we mainly focused on the domain-independence of the prediction models. However, the interpretability of models is a crucial and still open question which we would like to address in our future works. In particular, we think of using a teacher-student learning strategy, such as distillation [37], to transfer knowledge learned by a complex prediction model (teacher) into a simpler and

more interpretable one (student) which makes prediction almost as accurate as its complex counterpart. We can then use the simple model to study dependencies between the input data describing a user and the item predictions generated by the model.

Another challenge, especially for domains where implicit feedback is prevalent, is how to generate relevant recommendations by exploiting users' behavioral patterns. Existing approaches predict on which items the user will perform a target action and then consider these items as good recommendations. Even though these predictions can be accurate, the naive assumption that predicted items are good recommendations is not always reasonable. For example, implicit feedback models can suggest items which are obvious or can be autonomously discovered by the users. Thus, in our future works, we would like to focus more on the task of building useful and serendipitous recommendations from the predicted candidate items. To accomplish that goal, one can use item re-ranking techniques, such as calibration [47, 96], which can ensure that recommendations reflect various areas of interest of a user.

Finally, to help users to make better choices, one has to go beyond recommendation techniques and algorithms and consider human decision making processes. It is clear that an RS helps users to make decisions, but our understanding of its effect on item selection processes is still incomplete, and we need to connect RS research to psychology and decision making disciplines better. Further research in this area can help us to build better recommendation techniques that can effectively support users in making decisions.

To sum up, research on recommender systems and more specifically implicit feedback have come a long way. However, there are still a lot of research questions and challenges that are open for future explorations. We believe that this thesis will help to tackle at least some of them.

The source code of the models and experiments presented in this thesis is available at https://gitlab.inf.unibz.it/tural-gurbanov/mapm.

Bibliography

[1] Agrawal, R. and Srikant, R. (1994). 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. Morgan Kaufmann Publishers Inc.

[2] Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 3–14, Washington, DC, USA. IEEE Computer Society.

[3] Amatriain, X. and Basilico, J. (2015). *Recommender Systems Handbook*, chapter Recommender Systems in Industry: A Netflix Case Study, pages 385–419. Springer US, Boston, MA.

[4] Amatriain, X. and Pujol, J. M. (2015). *Data Mining Methods for Recommender Systems*, pages 227–262. Springer US, Boston, MA.

[5] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[6] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[7] Bonnin, G. and Jannach, D. (2014). Automated generation of music playlists: Survey and experiments. *ACM Comput. Surv.*, 47(2):26:1–26:35.

[8] Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[9] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

[10] Burke, R. (2007). The adaptive web. In Brusilovsky, P., Kobsa, A., and Nejdl, W., editors, *The Adaptive Web*, chapter Hybrid Web Recommender Systems, pages 377–408. Springer-Verlag, Berlin, Heidelberg.

[11] Chen, S., Xu, J., and Joachims, T. (2013). 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. ACM.

[12] Ching, W.-K., Huang, X., Ng, M. K., and Siu, T. K. (2013). *Markov Chains: Models, Algorithms and Applications*. Springer Publishing Company, Incorporated, 2nd edition.

[13] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

[14] Claypool, M., Le, P., Wased, M., and Brown, D. (2001). Implicit interest indicators. In *Proceedings of the 6th International Conference on Intelligent User Interfaces*, IUI '01, pages 33–40, New York, NY, USA. ACM.

[15] Covington, P., Adams, J., and Sargin, E. (2016). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 191–198, New York, NY, USA. ACM.

[16] De Boom, C., Agrawal, R., Hansen, S., Kumar, E., Yon, R., Chen, C.-W., Demeester, T., and Dhoedt, B. (2018). Large-scale user modeling with recurrent neural networks for music discovery on multiple time scales. *Multimedia Tools Appl.*, 77(12):15385–15407.

[17] Deshpande, M. and Karypis, G. (2004). Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177.

[18] Ekstrand, M. D., Harper, F. M., Willemsen, M. C., and Konstan, J. A. (2014). User perception of differences in recommender algorithms. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 161–168, New York, NY, USA. ACM.

[19] Elahi, M., Braunhofer, M., Gurbanov, T., and Ricci, F. (2019). *Collaborative Recommendations*, chapter User Preference Elicitation, Rating Sparsity and Cold Start, pages 253–294.

[20] Farris, P. W., Bendle, N. T., Pfeifer, P. E., and Reibstein, D. J. (2010). *Marketing Metrics: The Definitive Guide to Measuring Marketing Performance*. Wharton School Publishing, 2nd edition.

[21] Feng, S., Li, X., Zeng, Y., Cong, G., Chee, Y. M., and Yuan, Q. (2015). 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.

[22] Gasparic, M., Gurbanov, T., and Ricci, F. (2017a). Context-aware integrated development environment command recommender systems. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 688–693, Piscataway, NJ, USA. IEEE Press.

[23] Gasparic, M., Gurbanov, T., and Ricci, F. (2018). Improving integrated development environment commands knowledge with recommender systems. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*, ICSE-SEET '18, pages 88–97, New York, NY, USA. ACM.

[24] Gasparic, M., Janes, A., Ricci, F., Murphy, G. C., and Gurbanov, T. (2017b). A graphical user interface for presenting integrated development environment command recommendations: Design, evaluation, and implementation. *Information & Software Technology*, 92:236–255.

[25] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. The MIT Press.

[26] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *CoRR*, abs/1410.5401.

[27] Grbovic, M., Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V., and Sharp, D. (2015). 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. ACM.

[28] Gron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition.

[29] Gunawardana, A. and Shani, G. (2015). *Evaluating Recommender Systems*, pages 265–308. Springer US, Boston, MA.

[30] Gurbanov, T. and Ricci, F. (2017a). Action prediction models for recommender systems based on collaborative filtering and sequence mining hybridization. In *Proceedings of the Symposium on Applied Computing*, SAC '17, pages 1655–1661, New York, NY, USA. ACM.

[31] Gurbanov, T. and Ricci, F. (2017b). Exploiting multiple action types in recommender systems. In *Proceedings of the 8th Italian Information Retrieval Workshop, Lugano, Switzerland, June 05-07, 2017*, pages 72–75.

[32] Gurbanov, T., Ricci, F., and Ploner, M. (2016). Modeling and predicting user actions in recommender systems. In *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*, UMAP '16, pages 151–155, New York, NY, USA. ACM.

[33] He, H. and Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Trans. on Knowl. and Data Eng.*, 21:1263–1284.

[34] Hidasi, B. and Karatzoglou, A. (2018). Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, pages 843–852, New York, NY, USA. ACM.

[35] Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2015). Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939.

[36] Hidasi, B., Quadrana, M., Karatzoglou, A., and 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 '16, pages 241–248, New York, NY, USA. ACM.

[37] Hinton, G. E., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

[38] Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, Washington, DC, USA. IEEE Computer Society.

[39] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[40] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2014). *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated.

[41] Jannach, D. and Hegelich, K. (2009). A case study on the effectiveness of recommendations in the mobile internet. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 205–208, New York, NY, USA. ACM.

[42] Jannach, D., Lerche, L., and Jugovac, M. (2015). 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. ACM.

[43] Jannach, D., Lerche, L., and Zanker, M. (2018). *Recommending Based on Implicit Feedback*, pages 510–569. Springer International Publishing, Cham.

[44] Jannach, D. and Ludewig, M. (2017). When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 306–310, New York, NY, USA. ACM.

[45] Jawaheer, G., Weller, P., and Kostkova, P. (2014). Modeling user preferences in recommender systems: A classification framework for explicit and implicit user feedback. *ACM Trans. Interact. Intell. Syst.*, 4(2):8:1–8:26.

[46] Johnson, C. C. (2014). Logistic matrix factorization for implicit feedback data. *Advances in Neural Information Processing Systems*, 27.

[47] Jugovac, M., Jannach, D., and Lerche, L. (2017). Efficient optimization of multiple recommendation quality factors according to individual user tendencies. *Expert Systems with Applications*, 81:321 – 331.

[48] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[49] Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. (1997). Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87.

[50] Kordumova, S., Kostadinovska, I., Barbieri, M., Pronk, V., and Korst, J. (2010). Personalized implicit learning in a music recommender system. In *Proceedings of the 18th International Conference on User Modeling, Adaptation, and Personalization*, UMAP'10, pages 351–362, Berlin, Heidelberg. Springer-Verlag.

[51] Koren, Y. (2008). Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 426–434, New York, NY, USA. ACM.

[52] Koren, Y. and Bell, R. (2015). *Advances in Collaborative Filtering*, pages 77–118. Springer US, Boston, MA.

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

[54] Lee, W. S. and Liu, B. (2003). Learning with positive and unlabeled examples using weighted logistic regression. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 448–455. AAAI Press.

[55] Lerche, L. and Jannach, D. (2014). Using graded implicit feedback for bayesian personalized ranking. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 353–356, New York, NY, USA. ACM.

[56] Lerche, L., Jannach, D., and Ludewig, M. (2016). 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. ACM.

[57] Lin, C. H., Kamar, E., and Horvitz, E. (2014). Signals in the silence: Models of implicit feedback in a recommendation system for crowdsourcing. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pages 908–914. AAAI Press.

[58] Liu, S., Wu, Q., and Miao, C. (2018). Personalized recommendation considering secondary implicit feedback. In *2018 IEEE International Conference on Agents (ICA)*, pages 87–92.

[59] Liu, T.-Y. (2009). Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331.

[60] Liu, Y., Zhao, P., Sun, A., and Miao, C. (2015). A boosting algorithm for item recommendation with implicit feedback. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1792–1798. AAAI Press.

[61] Loni, B., Pagano, R., Larson, M., and Hanjalic, A. (2016). Bayesian personalized ranking with multi-channel user feedback. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 361–364, New York, NY, USA. ACM.

[62] Luong, M., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025.

[63] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[64] McFee, B. and Lanckriet, G. R. G. (2011). The natural language of playlists. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR 2011.

[65] Melville, P., Mooney, R. J., and Nagarajan, R. (2002). Content-boosted collaborative filtering for improved recommendations. In *Eighteenth National Conference on Artificial Intelligence*, pages 187–192, Menlo Park, CA, USA. American Association for Artificial Intelligence.

[66] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[67] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

[68] Mobasher, B., Dai, H., Luo, T., and Nakagawa, M. (2002). 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.

[69] Morita, M. and Shinoda, Y. (1994). Information filtering based on user behavior analysis and best match text retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 272–281, New York, NY, USA. Springer-Verlag New York, Inc.

[70] Nichols, D. M. (1997). Implicit rating and filtering. In *Proceedings of the 5th DELOS Workshop on Filtering and Collaborative Filtering*, pages 31–36.

[71] Ning, X., Desrosiers, C., and Karypis, G. (2015). *A Comprehensive Survey of Neighborhood-Based Recommendation Methods*, pages 37–76. Springer US, Boston, MA.

[72] Oard, D. and Kim, J. (1998). Implicit feedback for recommender systems. In *in Proceedings of the AAAI Workshop on Recommender Systems*, pages 81–83.

[73] Oard, D. W. and Kim, J. (2001). Modeling information content using observable behavior. In *Proceedings of the American Society for Information Science and Technology*.

[74] Olah, C. and Carter, S. (2016). Attention and augmented recurrent neural networks. *Distill*.

[75] O'Mahony, M., Hurley, N., Kushmerick, N., and Silvestre, G. (2004). Collaborative recommendation: A robustness analysis. *ACM Trans. Internet Technol.*, 4(4):344–377.

[76] Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R., Scholz, M., and Yang, Q. (2008). One-class collaborative filtering. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 502–511, Washington, DC, USA. IEEE Computer Society.

[77] Pan, W., Zhong, H., Xu, C., and Ming, Z. (2015). Adaptive bayesian personalized ranking for heterogeneous implicit feedbacks. *Knowledge-Based Systems*, 73:173 – 180.

[78] Parra, D. and Amatriain, X. (2011). Walk the talk: Analyzing the relation between implicit and explicit feedback for preference elicitation. In *Proceedings of the 19th International Conference on User Modeling, Adaption, and Personalization*, UMAP'11, pages 255–268, Berlin, Heidelberg. Springer-Verlag.

[79] Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop*, pages 39–42.

[80] Pizzato, L., Chung, T., Rej, T., Koprinska, I., Yacef, K., and Kay, J. (2010). Learning user preferences in online dating. In *In Proc. Preference Learning Workshop, ECML PKDD Conference*.

[81] Qiu, H., Liu, Y., Guo, G., Sun, Z., Zhang, J., and Nguyen, H. T. (2018). Bprh: Bayesian personalized ranking for heterogeneous implicit feedback. *Information Sciences*, 453:80 – 98.

[82] Quadrana, M., Cremonesi, P., and Jannach, D. (2018). Sequence-aware recommender systems. *ACM Comput. Surv.*, 51(4):66:1–66:36.

[83] Quadrana, M., Karatzoglou, A., Hidasi, B., and Cremonesi, P. (2017). Personalizing session-based recommendations with hierarchical recurrent neural networks. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 130–137, New York, NY, USA. ACM.

[84] Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009). 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. AUAI Press.

[85] Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. (2010). 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. ACM.

[86] Rendle, S. and Schmidt-Thieme, L. (2008). Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 251–258, New York, NY, USA. ACM.

[87] Ricci, F., Rokach, L., and Shapira, B., editors (2015a). *Recommender Systems Handbook*. Springer US, Boston, MA, 2nd edition.

[88] Ricci, F., Rokach, L., and Shapira, B. (2015b). *Recommender Systems Handbook*, chapter Recommender Systems: Introduction and Challenges, pages 1–34. Springer US, Boston, MA, 2nd edition.

[89] Shani, G., Heckerman, D., and Brafman, R. I. (2005). An mdp-based recommender system. *J. Mach. Learn. Res.*, 6:1265–1295.

[90] Shapira, B., Taieb-Maimon, M., and Moskowitz, A. (2006). Study of the usefulness of known and new implicit indicators and their optimal combination for accurate inference of users interests. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pages 1118–1119, New York, NY, USA. ACM.

[91] Shi, C., Liu, J., Zhang, Y., Hu, B., Liu, S., and Yu, P. S. (2018). Mfpr: A personalized ranking recommendation with multiple feedback. *Trans. Soc. Comput.*, 1(2):7:1–7:22.

[92] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550:354–359.

[93] Smirnova, E. and Vasile, F. (2017). Contextual sequence modeling for recommendation with recurrent neural networks. In *Proceedings of the 2nd Workshop on Deep Learning for Recommender Systems*, DLRS 2017, pages 2–9, New York, NY, USA. ACM.

[94] Song, Y., Elkahky, A. M., and He, X. (2016). Multi-rate deep learning for temporal recommendation. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 909–912, New York, NY, USA. ACM.

[95] Srebro, N. and Jaakkola, T. (2003). Weighted low-rank approximations. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 720–727. AAAI Press.

[96] Steck, H. (2018). Calibrated recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, pages 154–162, New York, NY, USA. ACM.

[97] Stevens, F. C. (1993). *Knowledge-based Assistance for Accessing Large, Poorly Structured Information Spaces*. PhD thesis, Boulder, CO, USA. UMI Order No. GAX93-20482.

[98] Sun, Y., Yuan, N. J., Xie, X., McDonald, K., and Zhang, R. (2016). Collaborative nowcasting for contextual recommendation. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 1407–1418, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[99] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

[100] Tan, Y. K., Xu, X., and Liu, Y. (2016). Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 17–22, New York, NY, USA. ACM.

[101] Tang, L., Long, B., Chen, B.-C., and Agarwal, D. (2016). An empirical study on recommendation with multiple types of feedback. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 283–292, New York, NY, USA. ACM.

[102] Twardowski, B. (2016). 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. ACM.

[103] Vasile, F., Smirnova, E., and Conneau, A. (2016). Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 225–232, New York, NY, USA. ACM.

[104] Wang, Y., Sun, H., and Zhang, R. (2014). *Web-Age Information Management: 15th International Conference, WAIM 2014, Macau, China, June 16-18, 2014. Proceedings*, chapter AdaMF:Adaptive Boosting Matrix Factorization for Recommender System, pages 43–54. Springer International Publishing, Cham.

[105] Zhang, Y., Dai, H., Xu, C., Feng, J., Wang, T., Bian, J., Wang, B., and Liu, T.-Y. (2014). 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.

[106] Zhou, K., Yang, S.-H., and Zha, H. (2011). Functional matrix factorizations for cold-start recommendation. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 315–324, New York, NY, USA. ACM.

[107] Zhu, H., Chang, D., Xu, Z., Zhang, P., Li, X., He, J., Li, H., Xu, J. W., and Gai, K. (2019). Joint optimization of tree-based index and deep model for recommender systems. *ArXiv*, abs/1902.07565.

[108] Zhu, H., Li, X., Zhang, P., Li, G., He, J., Li, H., and Gai, K. (2018). Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 1079–1088, New York, NY, USA. ACM.

[109] Zigoris, P. and Zhang, Y. (2006). Bayesian adaptive user profiling with explicit & implicit feedback. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, pages 397–404, New York, NY, USA. ACM.

Appendix A

Publications

This appendix provides the complete list of the author's publications. Each publication is supported by the description of the author's individual contribution.

A.1 Modeling and Predicting User Actions in Recommender Systems

> Gurbanov, T., Ricci, F., and Ploner, M. (2016). Modeling and predicting user actions in recommender systems. In *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*, UMAP '16, pages 151–155, New York, NY, USA. ACM

This work has been done in collaboration with the online video streaming service 7TV[1] which is a part of German mass media company P7S1[2]. The paper presents a general multi action prediction model for implicit feedback RSs that has been designed and implemented by the author of this thesis. The paper was published at the 24^{th} Conference on User Modeling, Adaptation and Personalization (UMAP'16).

[1] http://www.7tv.de/
[2] http://www.prosiebensat1.com

A.2 Action Prediction Models for Recommender Systems Based on Collaborative Filtering and Sequence Mining Hybridization

> Gurbanov, T. and Ricci, F. (2017a). Action prediction models for recommender systems based on collaborative filtering and sequence mining hybridization. In *Proceedings of the Symposium on Applied Computing*, SAC '17, pages 1655–1661, New York, NY, USA. ACM

In this paper, a generic hybrid action prediction model leveraging observations of actions of different types has been studied. The model takes into account the information about the time delays between actions and the ordering of actions. The author of this thesis designed, implemented, and evaluated the presented model. The paper was accepted for the track on RSs of the 32^{nd} ACM Symposium on Applied Computing (SAC'17).

A.3 Context-Aware Integrated Development Environment Command Recommender Systems

> Gasparic, M., Gurbanov, T., and Ricci, F. (2017a). Context-aware integrated development environment command recommender systems. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 688–693, Piscataway, NJ, USA. IEEE Press

The research is the joint work with Marko Gašparič. The author of this thesis contributed to the design and implemented the context-aware IDE command RS and a new evaluation metric that is specifically measuring the usefulness of contextual recommendations. The paper was published at the 32^{nd} IEEE/ACM International Conference on Automated Software Engineering (ASE'17).

A.4 Exploiting Multiple Action Types in Recommender Systems

> Gurbanov, T. and Ricci, F. (2017b). Exploiting multiple action types in recommender systems. In *Proceedings of the 8th Italian Information Retrieval Workshop, Lugano, Switzerland, June 05-07, 2017*, pages 72–75

This work is an extended abstract where the research questions covered in this thesis have been introduced for the first time. The paper highlights the author's research in the field of implicit feedback RSs leveraging information about multiple action types. It was accepted for the 8^{th} Italian Information Retrieval Workshop (IIR'17).

A.5 A Graphical User Interface for Presenting Integrated Development Environment Command Recommendations: Design, Evaluation, and Implementation

> Gasparic, M., Janes, A., Ricci, F., Murphy, G. C., and Gurbanov, T. (2017b). A graphical user interface for presenting integrated development environment command recommendations: Design, evaluation, and implementation. *Information & Software Technology*, 92:236–255

This publication is a continuation of the joint work with Marko Gašparič on IDE command RSs. The paper describes and evaluates a novel design of a graphical user interface to recommend commands within an IDE. The interface contains a description of the suggested command, an explanation of why the command is recommended, and a command usage example. The author of this thesis designed and implemented an IDE command RS and an explanation tool for the recommendations. The paper was published in the "Information and Software Technology" journal.

A.6 Improving Integrated Development Environment Commands Knowledge with Recommender Systems

> Gasparic, M., Gurbanov, T., and Ricci, F. (2018). Improving integrated development environment commands knowledge with recommender systems. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*, ICSE-SEET '18, pages 88–97, New York, NY, USA. ACM

This work is a necessary complement to the previous research on IDE command RSs. It presents a year-long user study in which first-year computer science students have interacted with the complete IDE command RS. The system integrated data collection tools, three different algorithms, and a GUI proposed in the previous research. The paper addresses a set of research questions that are focused on the overall acceptance of the RS in real-life settings.

The paper was published at the 40^{th} International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET'18).

A.7 User Preference Elicitation, Rating Sparsity and Cold Start

> Elahi, M., Braunhofer, M., Gurbanov, T., and Ricci, F. (2019). *Collaborative Recommendations*, chapter User Preference Elicitation, Rating Sparsity and Cold Start, pages 253–294

This book chapter is the joint work with Mehdi Elahi and Matthias Braunhofer. It discusses the cold start problem and provides a comprehensive description of techniques that have been proposed to address this problem. The chapter surveys algorithmic solutions, provides a list of publicly available resources (e.g., software frameworks and datasets), and offers a set of practical guidelines that can be adopted by researchers and practitioners. The chapter was published in the "Collaborative Recommendations" book.