



POLITECNICO DI MILANO  
DIPARTIMENTO DI ELETTRONICA, INFORMAZIONE E BIOINGEGNERIA  
DOCTORAL PROGRAMME IN COMPUTER SCIENCE

---

## CONTEXT-DRIVEN RECOMMENDER SYSTEMS

Doctoral Dissertation of:  
**Roberto Pagano**

Supervisor:  
**Prof. Paolo Cremonesi**

Tutor:  
**Prof. Simone Garatti**

The Chair of the Doctoral Program:  
**Prof. Andrea Bonarini**

2016 – cycle XXIX



---

---

## Abstract

---

**R**ECOMMENDER systems always relied on personalization and context has only been used to improve personalization performance. However algorithms that decouple the context exploitation from personalization were not fully explored. This thesis presents *Context-Driven Recommender Systems*, a new family of recommendation algorithms that *personalize on user's intent and situation*, exploring their applicability in five application domains. This work shows that leveraging contextual information in these domains helps in addressing the challenges that they pose to traditional recommendation approaches and improves the performance. This new paradigm improves serendipity and can “pop” the filter bubble, can provide recommendation to unregistered or unlogged users, does not suffer from cold start problem and respects the privacy of users.



---

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	From context-aware to context-driven recommender systems . . . . .	1
1.2	Modeling context in new application domains . . . . .	4
<b>2</b>	<b>State Of The Art</b>	<b>9</b>
<b>3</b>	<b>Music</b>	<b>13</b>
3.1	Problem Definition . . . . .	14
3.2	Background . . . . .	15
3.3	Implicit playlist recommender (IPR) . . . . .	15
3.4	Experiment Setup . . . . .	16
3.4.1	Dataset . . . . .	16
3.4.2	Compared Methods . . . . .	21
3.4.3	Evaluation Methodology . . . . .	22
3.4.4	Settings . . . . .	22
3.5	Results . . . . .	23
3.5.1	Computing time . . . . .	25
3.6	Discussion . . . . .	25
<b>4</b>	<b>Jobs</b>	<b>27</b>
4.1	Problem Definition . . . . .	27
4.2	Background . . . . .	29
4.2.1	Related work . . . . .	31
4.3	Multi-Stack Ensemble and Multi-Channel Sampling . . . . .	32
4.3.1	Multi-Stack Ensemble . . . . .	32
4.3.2	Multi-Channel BPR-FM . . . . .	38
4.4	Experiment Setup . . . . .	42
4.4.1	XING Dataset . . . . .	42
4.4.2	Experiment 1: The 2016 RecSys Challenge . . . . .	43
4.4.3	Experiment 2: Multi-Channel BPR . . . . .	44
4.5	Results . . . . .	44

## Contents

---

4.5.1	Experiment 1: The 2016 RecSys Challenge . . . . .	44
4.5.2	Experiment 2: Multi-Feedback BPR . . . . .	45
4.6	Discussion . . . . .	48
<b>5</b>	<b>Linear TV Program Recommendation</b>	<b>51</b>
5.1	Problem Definition . . . . .	51
5.1.1	Notation . . . . .	53
5.2	Background . . . . .	54
5.3	Discrete and Smoothed Contextual Recommender . . . . .	56
5.3.1	Discrete Contextual Recommender . . . . .	56
5.3.2	Smoothed Contextual Recommender . . . . .	57
5.4	Experiment Setup . . . . .	59
5.4.1	Dataset . . . . .	59
5.4.2	Experiment 1: Discrete Contextual Recommender . . . . .	62
5.4.3	Experiment 2: Smoothed Contextual Recommender . . . . .	64
5.5	Results . . . . .	65
5.5.1	Experiment 1: Discrete Contextual Recommender . . . . .	65
5.5.2	Experiment 2: Smoothed Contextual Recommender . . . . .	68
5.6	Discussion . . . . .	69
<b>6</b>	<b>Linear TV Audience Prediction</b>	<b>71</b>
6.1	Problem Definition . . . . .	72
6.2	Background . . . . .	73
6.3	Dynamic Models . . . . .	74
6.4	Experiment Setup . . . . .	77
6.4.1	Compared Methods . . . . .	77
6.4.2	Settings . . . . .	78
6.5	Results . . . . .	79
6.6	Discussion . . . . .	80
<b>7</b>	<b>E-Tourism</b>	<b>83</b>
7.1	Problem Definition . . . . .	84
7.2	Background . . . . .	84
7.3	Context-Driven Available Hotel Recommender . . . . .	84
7.4	Experiment Setup . . . . .	85
7.4.1	Compared Methods . . . . .	86
7.5	Results . . . . .	87
7.6	Discussion . . . . .	88
<b>8</b>	<b>Mobile Apps</b>	<b>91</b>
8.1	Introduction . . . . .	91
8.2	Background . . . . .	93
8.2.1	Mobile application installation scenario . . . . .	93
8.2.2	Mobile application usage scenario . . . . .	97
8.2.3	Datasets . . . . .	100
8.3	Context-driven hybrid usage prediction algorithm . . . . .	102
8.4	Experiment setup . . . . .	104

8.4.1 Dataset . . . . .	104
8.4.2 Evaluation Methodology . . . . .	107
8.5 Results . . . . .	107
8.6 Discussion . . . . .	107
<b>9 Discussion and Conclusions</b>	<b>113</b>
<b>Bibliography</b>	<b>116</b>





---

# CHAPTER 1

---

## Introduction

---

### 1.1 From context-aware to context-driven recommender systems

---

The recommender systems community has always tried to use additional information (or evidence) beyond information about users and items [2]. Context is an example of evidence successfully exploited in many recommendation tasks.

We define context as the combination of what is going on around the user in the moment of the interaction with the system (*situation*) and on what she is trying to accomplish (*intent*). Contextual variables are the *signals* from which context can be inferred.

This thesis describes a new conceptualization of recommender systems: context-driven recommender systems [97]. Context-driven recommender systems (CDRS) have their roots in context-aware recommender systems, but they decouple the context exploitation from personalization. Rather than personalize, i.e. tailor recommendation to the individual, context-driven algorithms aim to contextualize, i.e. to customize (or personalize) on the user intent and situation. In the context-driven paradigm the subject of the personalization changes: instead of personalizing on the user, it personalizes on the context. In this way, the user is split into a collection of different dynamics, each one reflecting her different intents and situations. By the mean of this break-down, context-driven algorithms are able to model a more fine-grained similarity: the similarity which arises from comparing contexts, not simply users or items.

In recent years we have seen that information retrieval (IR) and recommender systems came closer each other. In the IR community, relevance has long been acknowledged as situational, e.g., dynamic, multidimensional, and dependent on user goals and situation [26], but this concept has not really been explored by the recommender system community.

Context-driven recommender systems differ from context-aware recommender sys-

tems for their focus on context rather than preferences. While context-aware recommender systems are tailored to the user preferences, relegating the context as a side information, context-driven recommender systems use the context as their main source of information, and past user preferences may or even may not be taken in consideration. When context-driven recommender systems make use of user preferences, these preferences are not considered a property of the user, but a property of the context in which they appear. In other words context-driven recommender systems personalize on intents and situations, instead of personalizing on users. Context-driven recommender systems (CDRS) try to elicit the goal of the user: usually the goal cannot be accomplished in a single atomic step, but it requires several steps to be performed. Under this consideration, two peculiarities of CDRS arise: *(i)* they are a decision support system, because they help the user to accomplish a task and *(ii)* current interactions of the user with the system (i.e. session), preferences in similar contexts, and other contextual signals are the key pieces of information to elicit the user goal.

CDRS use the contextual state of the user instead of the user profile as their main input source. The rationale is that context is relative to the condition of the user and may lead to several goals. As the user makes decisions, the user goal becomes clearer and clearer from the system's point of view. Two users that share the same context (or conditions) and make the same decisions are considered as equal from a context-driven point of view.

The research community has just started to tackle the *context-driven* nature of human tastes: [100, 58, 97]. Recent works [17, 84, 58] have designed algorithms mainly focused on the current context of the user: this is partly due to the exploration of new application domains that *(i)* exhibit a strong contextual item consumption process and *(ii)* need to provide recommendations to unregistered or non logged users (cold start users). In the second case, the only information that can be exploited is the session, i.e. the current interactions of the user with the system (e.g. search query, pages visited and items browsed). In many application domains user tastes are not static, but they drift with time and more generally with the context while the latter can change between sessions and even within the same session.

Several factors make context-driven recommendation feasible: a huge amount of contextual data is generated by smartphones and Internet of Things devices and then stored on the cloud, and big data technologies make the computational power to process it finally available. Moreover, privacy concerns are growing and the storage of personal information is becoming more and more difficult. So algorithms that do not make use of this information can be compliant with the new privacy regulations.

The availability of such amount of data regarding the context paves the way for a revolution of recommender systems: the shift from the static nature of the *personalization*, where user preferences are carved into the stones of their profile, reinforcing the filter bubble effect [103] and thus not allowing for the discovery of new items and interests, to the *contextualization*, where the user taste is drifting according to the goal to be accomplished.

Many application domains call for recommendations that are actually *driven* by the context, thus relaxing the constraint that personalization involves recommendation for specific individuals. In the context-driven paradigm each user is not (only) identified by her id, but by her contextual states, thus dissociating users from their historical

## 1.1. From context-aware to context-driven recommender systems

behavior. In this way, users have more room to develop and are not tied to their past actions. Context-driven recommender systems implicitly assume that *people have more in common with other people in the same situation, or with the same goals, than they do with past versions of themselves.*

In the linear TV domain, for example, the user may switch channel multiple times or even the person controlling the TV can change very frequently. In this domain the tastes are driven by the channel and the hour of the day, rather than the intrinsic quality of the aired program. In the music domain, for example, the taste is driven by the task the user is performing (e.g. working or jogging), the mood, the device used, and the trend of the last played songs [56]. In the E-tourism domain the driving factors for the particular choice of the hotel or restaurant are the weather, the people that are with the user and the time. In the flight booking domain the period of the year or the occurrence of particular events are crucial for users' booking intentions.

Together, these new resources and new insights make *now* the right moment to launch consolidated effort towards context-driven recommendation.

This thesis does not claim that personalization is not important or not useful, but presents a new point of view that is still compatible with it. In order to understand and overcome the limits of personalization, the factors that gave birth to it should be considered. Personalized products are nothing new. Before the industrial revolution and modern mass production<sup>1</sup>, most of the more expensive products were commissioned and tradesmen took great care to take their customers' needs into consideration. With the rise of the digital era and recommender systems, companies have started to add personalization in their content delivery. Since the start of the Netflix Prize, the research in recommender systems and on personalization has been energized. However the Netflix Prize heavily biased the conception of recommender systems from the very beginning. First of all user and item ids were the only information available (or at least the one Netflix provided for the Prize). Besides Netflix, many marketing studies tell us that users are more happy with recommendations "tailored for them"<sup>2</sup>. Moreover adding personalization enables companies to deliver a diversified set of items to each user, thus increasing the coverage of the item catalog. Since the introduction of context-aware recommender systems, context was plugged in the personalized recommender with success [10, 112, 101, 100, 13, 27, 58]. However this thesis claims that context can be usefully exploited by moving one step forward from personalization. Although many companies have already understood the potential of context and are employing context-aware recommenders, they did not move away from personalization<sup>3</sup>.

Modeling user needs as a static and as an invariant property of an individual does not give room to her tastes to develop, by constantly keeping her in a "filter bubble" on which the individual has no control [103]. Although at first glance this can be seen as opposed to the personalization, in fact it is its natural evolution and extension, allowing a finer grain control by considering not only the individual, but also the factors that led that individual to interact with the system. In this way, rather than considering an individual as a single entity, a context-driven algorithm models her as dynamic and

<sup>1</sup><http://yfsmagazine.com/2015/05/20/have%2Dit%2Dyour%2Dway%2Dthe%2Drise%2Dof%2Donline%2Dpersonalization%2Dtrends/>

<sup>2</sup><https://www.linkedin.com/pulse/personalization%2Dpillar%2Dmobile%2Duser%2Dexperience%2Dalon%2Deven>

<sup>3</sup><http://www.retailtouchpoints.com/features/special%2Dreports/retailers%2Dseek%2Dinnovation%2Din%2Dpersonalization>

multifaceted, with preferences and patterns that vary according to the circumstances.

This new conceptualization tries to overcome the limits of some assumptions implicitly made by recommender systems, i.e. the Immutable Preference paradigm (ImP). From the very beginning, recommender systems assumed that users goals, needs and tastes do not evolve. ImP also assumes that the item catalog is static, without accounting for trendiness, seasonality and dynamicity of items. Currently some tweaks are used in order to account for these issues, like applying a decay to give more importance to recent preferences or they adjust weights to account seasonality or trends. Some underlying challenges of recommender systems are driven to the extreme with this new conceptualization: the sparsity problem is even more evident, since context-driven use a context state as target. Moreover the cold-start problem is becoming a *continous cold-start* problem due to the dynamicity of users and items. Context-driven recommender systems reject the assumption of ImP in their algorithmic fabric, rather than as an afterthought. A context-driven algorithm borrows some key features from collaborative, content, and context-aware (and session-aware) algorithms. It is collaborative in the sense that collaboration happens among contextual states instead of users. It is content-based, since the content of the item can be plugged in as another contextual variable to refine the system's understanding of the user context. It is context-aware, since it does make use of contextual variables.

Context-driven paradigm is the common theoretical ground for many disparate strands in the recommender systems community. Besides context-aware recommender systems, other research branches can be framed as context-driven problems such as session-based recommender systems, which model context as a series of user interactions carried out within a session. Stream based recommendation and content based recommendation share many features with context-driven recommendation, such as the ability to tackle cold start and sparse user interactions. By exploring the commonalities between a range of application domains and by discussing the potential of existing algorithms and the challenges that still remains open, we aim to encourage the development of this new breed of algorithms, specifically suited for context-driven recommendation.

Research community has welcomed the context-driven paradigm. The reference work for this new conceptualization is [97].

### 1.2 Modeling context in new application domains

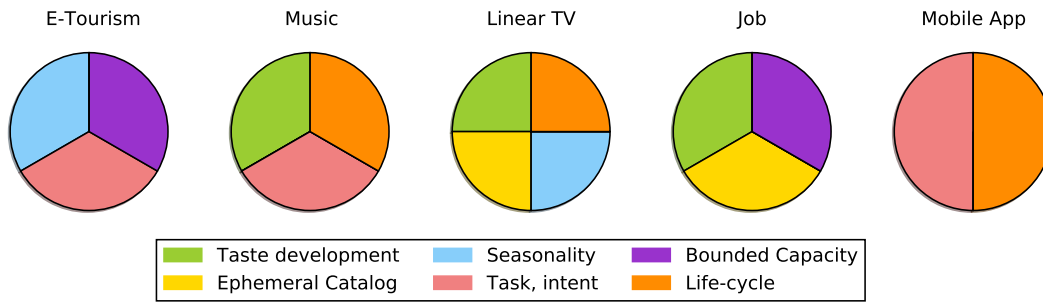
---

This research was carried out with the objective of finding the best way to exploit the contextual information in recommender systems. The research question that drives this thesis can be phrased as:

*Given an application domain and the challenges that it poses to recommender systems, which is the best way to leverage the context to overcome the limits of state-of-the-art recommendation algorithms?*

The five domains examined in this thesis were chosen for their contextual potential, i.e. the potential usefulness of the context in improving recommendation algorithm. Figure 1.1 depicts the challenges of the investigated application domains where ImP assumption fails to hold. For example, the ephemeral catalog property breaks the assumption on the steadiness of the item catalog in the linear TV and Job domains. New

## 1.2. Modeling context in new application domains



**Figure 1.1:** Application domains with challenges for context-driven recommendation

TV shows or jobs are added to the catalog very frequently as well as TV programs will not be aired anymore or jobs that are published for a long time lose their effectiveness. The same holds for seasonality in TV and E-tourism domains: many TV shows or hotels or flights are very much dependent on the season. For example very few people go to ski facilities when there is no snow to ski on, or go to the sea in the winter season. For TV, many programs are aired in a particular season of every year.

The taste development is another natural human condition that breaks the ImP assumption. Human tastes naturally evolve and recommenders should be able to cope with it. The same person before or after graduating, can look for different kinds of jobs (e.g. a part time job during studies and a well paid full time job that meets her aspirations after obtaining a degree). The same holds for music or TV: the same user may change consumption patterns over time and situation.

The availability of items is another issue that was not present in the very first application domain in which recommender systems were applied, i.e. *digital* movies. However there are different domains where items cannot be consumed an unlimited number of times: this is the case of E-tourism, where hotels or flights have a maximum capacity that limits their availability, or jobs, where each job request can be met only by a limited number of individuals.

The life-cycle property is relative to the item consumption curve in time: many items are very popular for a limited time, decreasing their appeal after this period. This property holds in the music domain, where the very popular songs start to fade out after being on the top of the charts for some time. The same happens for TV, where the first episode of a new show or series is very popular. This is also true for the app domain: many apps, especially games, are installed and uninstalled very frequently to give room to newer and fresher apps.

Finally, the intent or task makes preferences to change or to be inconsistent with themselves according to situation, thus breaking the concept on which ImP builds upon, i.e. preferences depend on user and item. There are domains where the task or the intent of the user is far more important than her past preferences. In the music domain, for example, the user might want to work out, so she might enjoy music that gives strength. Or the user might want to work, so she might enjoy relaxing and concentrating music. The task is even more important in the app usage prediction: if the user's intent is to look for a restaurant to have dinner, she may open a totally different set of apps with respect to the set of apps used for reading news.

This thesis investigates some of these challenges on these domains and shows that

each domain has some underlying preference model. In the early recommender systems years, the research focused on the user and item, so models have been developed to embody the preference model in the user and item variables because they were the only information available. But what if the underlying preference model does not depend much on the user and the item, but instead on some contextual variables that we previously did not take into account? Let us make an example: suppose that a user, Bob, a single businessman, has to travel for work to San Francisco when most of the hotels are fully booked. So he starts browsing the available hotels and clicks on some of them. Let us also suppose that another user, Alice, is looking to visit San Francisco for vacation with her family and clicks on the same hotels clicked by Bob, because of the limited availability. In this case any non-context aware recommender system, either collaborative or content-based, would recommend the same things to Bob and Alice. This can be good for the purpose of the trip to San Francisco, but the similarity between the two users has to expire somehow after that. There is an underlying reason that led the two users to click on the same hotels: they were the only available ones. Any recommender system that only exploits user and item ids cannot capture this concept. Furthermore it will model the two users as very close to each other when they are quite different in reality. In fact, the reasons that led the two users to click on the same hotels were basically depending on the *intent*, e.g. travel to San Francisco, and *situation*, e.g. most hotels were fully booked.

A context-driven recommender system, instead, should be able to capture user's intent and situation because it is *designed* to do so. Besides the user and item ids, there are a lot of contextual *signals* that they can exploit: first of all the availability should be taken into account. Second, the intent should be inferred by looking at the session of browsed hotels. Third, their similarity is only relative to the contextual situation "trip to San Francisco" and will not hold anymore after this situation. Moreover such an algorithm will be able to suggest the same items to other users after having inferred their intent and understood that the situation is similar.

In this example the underlying preference model cannot be really captured by looking at the user and item, but instead by looking at other contextual *signals*. In the hotel booking domain, for example, the preference model can be summarized with the following sentence "a user is looking for a hotel that has good reviews, is close to the destination, is available, meets the user's financial requirements and is appropriate for the people the user is traveling with".

**Thesis Structure** Chapter 2 presents the main related work on context-aware and context-driven recommender systems. Other domain-specific works are discussed in the other chapters. Chapter 3 explores the challenges that context poses to the Music domain and to the playlist recommendation setting in particular. Chapter 4 shows the peculiarities of the domain of job recommendation and presents our approach to The 2016 Rec-Sys Challenge as well as a context-driven sampling methodology. Chapter 5 deeply explores the benefits of context in the Linear TV program recommendation domain. Chapters 6 and 7 present problems having a strong dependency on context and where the user has a marginal role. In particular, Chapter 6 investigates on how to use context in the TV audience prediction problem while Chapter 7 presents a follow-up work to [33] showing that taking into account context improves the accuracy also in the E-

## **1.2. Modeling context in new application domains**

---

Tourism domain. Chapter 8 presents some preliminary work in the field of mobile app recommendation with a focus on the use of context. Finally, Chapter 9 draws some conclusions from the work presented in this thesis.





---

## CHAPTER 2

---

### State Of The Art

---

The context-driven paradigm is very novel and literature lacks of prior work on recommender systems only exploiting context without making use of personalization. In this chapter we present the main points relative to the way context is used in context-aware recommender systems which can be seen as the direct progenitor of context-driven recommender systems. In the next chapters we will present the related works on a per-domain basis, describing the state of the art of the examined domains more in detail.

Dourish in [47] defines two views on context: *representational* and *interactional*. In the first view context is seen as a representation problem, in which context is identifiable by a predefined set of observable attributes. This view makes the following assumptions: (i) context is a form of information and, so, is something that *can be known*; (ii) context is *delineable*, so we can, for some set of applications, define what counts as the context of the activities the main application supports; (iii) context is *stable*, so the elements of a context representation do not vary from instance to instance in a particular application; (iv) *context and activities are separable*, so that an activity happens within a context. The interactional view, instead, argues that (i) context is a *relational property* between objects or activities, so something may be or may not be contextually relevant to the particular activity; (ii) the scope of contextual features is *defined dynamically*; (iii) context is an *occasioned property* relative to each occasion of activity or action; (iv) context *arises from the activity*. In other words this view assumes a bidirectional relationship between context and activities. This distinction reflects the essential difference between the positivist and the phenomenological positions. From a positivist perspective (representational view), context is a stable feature of the world, independent of the actions of individuals. From a phenomenological perspective (interactional view), contextuality comes about only when it is mutually recognized by the parties to some interaction, drawing on their everyday, cultural, common-sense understandings

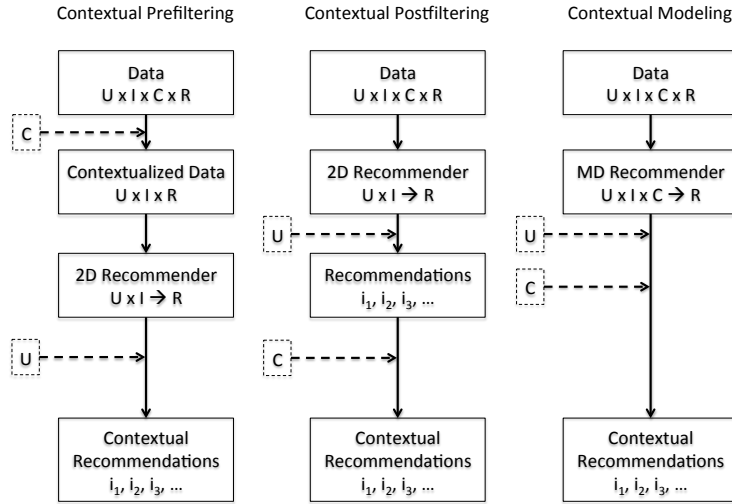


Figure 2.1: Prefiltering, postfiltering and contextual modeling for context-aware recommender systems

of the nature of the social world.

We could argue that our context definition builds on the interactional view and incorporates the representational view. It incorporates the representational view because we are clearly defining what context is: the combination of situation and intent, but this definition is not static nor delineable. We agree with the assumptions of the interactional view, but simply stating that every information that can be used to understand the situation and the intent of a user in a particular application setting is valuable to represent the context.

The exploitation of context for improving the quality of recommendation has begun with context-aware recommender system. The most comprehensive categorization of such algorithms has been described by Adomavivius and Tuzhilin in [2]. Context-aware recommender systems fall into three types: *contextual pre-filtering*, where context drives data selection; *contextual post-filtering*, where context is used to filter recommendations once they have been computed using a traditional approach; and *contextual modeling*, where context is integrated directly into the model. Figure 2.1 presents the three approaches. The input data is represented as a tuple  $U \times I \times C \times R$  containing user ( $U$ ), item ( $I$ ), context ( $C$ ) and rating ( $R$ ). The pre-filtering approach filters the data using one or more contextual variables. Then the data is fed to a classical recommender system. 2D stands for two-dimensional, since it uses only user and item information. In the post-filtering approach the data is fed to a 2D recommender ignoring the contextual variables and then the recommendation list is refined using the contextual variables. Contextual modeling, instead, makes use of the whole data to train a multi-dimensional recommender.

Pre and Post filtering approaches can leverage traditional recommendation algorithms while modifying either the input or the output of such algorithms. Contextual modeling, instead, requires new models that take into account the contextual variables during the learning phase. An example of contextual pre-filtering is the so-called *user micro-profile*, in which a single user is represented by a hierarchy of possibly overlap-

---

ping contextual profiles [10]. As for the post-filtering approach, there are two possible choices: (i) removing recommendations that are irrelevant to the context and (ii) adjusting the ranking of the recommendation list. An example of post-filtering in the music domain is described in [56]. The choice to use pre or post filtering approaches is not trivial: in their experimental evaluation, Panniello *et al.* [101] found that the choice of a pre-filtering or post-filtering strategy depends on the particular recommendation problem.

Another approach on modeling context is the user or item splitting. This approach creates fictitious users (items) by splitting them according to the different contexts in which they interacted (they have been interacted within). For the item splitting approach, the assumption is that the nature of an item, from the user’s point of view, may change in different contextual conditions (values of contextual variables) [12]. For example the same restaurant can be considered as two different restaurants if it is summer time or winter time, or if the weather is sunny or rainy. For the user splitting approach, it may be useful to consider one user as multiple users, if he or she demonstrates significantly different preferences in different contexts [10]. There are also approaches that split both the users and the items according to the context. An example of this approach is simply an application of item splitting followed by user splitting on the resulting output [144].

One of the most successful methods for integrating the contextual information into the model is the tensor factorization. Tensor factorization extends the matrix factorization into a higher dimensional matrix (i.e. a tensor) having contextual variables as additional dimensions. In this way, instead of creating two latent matrices, containing the user and the item latent factors, tensor factorization creates a latent matrix for each dimension, i.e. one for the user, one for the items and one for each contextual variable. The authors of [60] describe a method for applying the candecomp/parafac tensor factorization to implicit feedback datasets. Another notable example of contextual modeling are the Factorization Machines (FM) [112]. It has been proven that FMs have similar properties to tensor factorization. Conceptually, FMs horizontally stack the one-hot-encoding of user, item and contextual features for each user item interaction in a single training example and learn latent representations for both each feature and for the product of any pair of features (second level interactions). We point to Section 4.3.2 for an in depth analysis of this algorithm.

The idea of recommending to contextual states instead of users is not new. Some signals of this trend can be even traced back in the definition of Bayesian Personalized Ranking, one of the most used Collaborative Filtering methods. In [110] the authors change the domain of the predicted rating function, from  $\hat{r} : U \times I$  in the original definition [111], i.e. the predicted rating depends on the user and the item, to  $\hat{r} : C \times I$  in the improved version [110], i.e. the predicted rating depends on a context  $C$  (in which also the user can be included) and on an item  $I$ . However the implications of such formulation were not deeply explored by the recommender system community.

More recently, the best paper at ACM RecSys 2015 [84] propose a recommendation algorithm for points of interest that is able to tackle extreme cold-start situations by exploiting context. This can be considered an example of context-driven algorithm.

Session-based algorithms deserve a separate dissertation. These techniques still belong to the *contextual-modeling* category of context-aware recommender systems, but

they exploit the sequence of items the user interacted with as contextual variable. By using the session to infer the user *intent* and *situation*, an algorithm can learn relationships and patterns that cannot be exploited by traditional recommendation techniques. Turrin et al. [126] use recurrent sessions as implicit playlists in the music domain. Recurrent Neural Networks are successfully used to model sessions in the webshop and user generated content domains [58, 59].

The first conceptualization of context-driven algorithms was published in RecSys 2016 and constitutes the reference work that attempts to unify the different and fragmented research paths that use context as primary source of information for recommendation into a single theoretical formulation [97].

We point to the next chapters for an in depth background on the way context is used in a particular application domain.

---

## CHAPTER 3

---

### Music

---

In this chapter we address the challenges – task, intent, taste development and item life-cycle – that break the ImP assumption in the music domain, described in Figure 1.1. Music listening behaviour is very much dependent on the task the user is performing: if the user is running, she may want to listen to energetic music. If the user, instead, wants to focus, she might enjoy chill-out or classical music. Taste development make the users’ tastes drift with time, thus making the user modeling very difficult. Moreover the item life-cycle causes the songs to have a trendiness factor, making them particularly appealing for some limited time, fading out afterwards. The particular problem addressed is the playlist recommendation that perfectly fits in this scenario and presents great opportunities to context-driven algorithms. A context-driven algorithm assumes that a user is a collection of music tastes that vary according to the context. The challenge for such an algorithm is two-fold: first it has to distinguish between meaningful listening patterns and noise and, second, it has to understand which is the current taste of the user in order to perform the recommendations. We overcome the aforementioned challenges by modeling the user on the fly in the recommendation phase, by looking at the current listening session. In this way the past behaviour of the user does not influence the recommendation, giving users the possibility to actually “pop” the filter bubble. A particular attention has been posed on the scalability of the algorithm, since in the music domain the item set is very large with respect to other domains (e.g. movies). Moreover the number of interactions of the user with the system is also much higher because an active user can listen to dozens of songs per day. Our contribution in this field is two-fold: we create the largest music dataset that is publicly available today - *30Music* dataset [127]- and we propose an innovative context-driven approach - namely the *Implicit Playlist Recommender (IPR)* [126].

The main innovative aspects of the *30Music* dataset with respect to the existing

public datasets are:

- play events are organized into listening *sessions*, that constitute our only contextual information;
- the dataset contains user-generated playlists;
- the dataset contains both implicit play events and explicit user ratings, i.e., preferred tracks (positive feedback);
- whenever a user plays a track from a playlist, the play event is tagged.

The main innovations introduced by IPR are:

- considering recurrent listening sessions as playlists
- no user profile modeling, so to focus attention on the current listening session, emphasizing the importance of user intent and task
- the scalability of the algorithm makes it suitable for large scale setting

### 3.1 Problem Definition

---

Music recommender systems propose interesting music to a specific user. Differently from many other popular recommendation domains - such as video - items in the music domain are very short (few minutes) and are not consumed atomically, but usually as a bundle of songs.

It is important noting that recommending a group of songs has very little to share with recommending the top-N relevant songs to a user. In accordance with the definition proposed by Cunningham et. al [41], we focus on *music playlist recommendations*. A playlist is a group of songs relevant in a certain context, where the strict listening order does not necessarily matter. The user context strongly affects the music playlist consumption. Unfortunately, contextual information is usually missing and difficult to measure.

We overcome such limitation by using the user current listening session as a proxy to the user context (in particular to his intent). In fact, regardless of the user global preferences (e.g., preferred music genres), whether he starts listening to a certain kind of songs, he is indirectly expressing a strong, temporary preference. Consequently, the next song to suggest to such user has to be somehow aligned (e.g., similar genre, same theme, etc.) with the tracks listened in the current session. User listening sessions indeed form a rich and reliable set of implicit playlists that can be exploited to recommend new tracks relevant to the current listening session. The main steps of this solution consist of (i) identifying the user listening sessions from the atomic play events to create the implicit playlists and (ii) matching the user's current listening session to find the relevant implicit playlists, in accordance to a defined similarity metric (see Section 3.3).

As described in Section 3.4, we processed all the collected play events of 45K users during one year (31M events) on a 4.5M-item catalog, defining 2.7M sessions. Therefore, for each recommendation, the current session of the target user has to be compared with all the extracted implicit playlists. The best matching playlists are aggregated in order to identify the tracks to recommend.

## 3.2 Background

Music recommender systems are typically implemented to suggest to users either (i) top-n recommendations, i.e., a limited set of tracks, albums, or artists [116] that are believed to be appealing for them [77] or (ii) an ordered list of tracks or artist (i.e., playlist recommendations) [7, 24].

In this chapter we focus on presenting to a user a sequence of tracks, referred to as *playlist*. An updated list of solutions can be found in Bonnin and Jannach’s survey [25]. Among the top performing algorithms, they mention Same Artist Greatest Hits (SAGH) and Collocated Artist Greatest Hits (CAGH), which we have used as baselines (see Section 3.4.2).

Ragno et al. [106] started from the consideration that the more frequently two tracks are played one after the other, the more similar they are. They proposed to use “editorial playlists” to build a graph where every node is a track and every weighted edge represents the similarity ratio among a couple of nodes: the recommendation is generated through a Random Walk in this graph.

A more recent work is presented by Dzuba and Bugaychenko [49]: they mined the user listening behavior to identify frequent patterns and build playlists, further optimized with respect to track diversity. Hariri et al. [56] tried a similar approach focusing on the next-artist recommendation problem: they build a graph and used a Markov Chain model to recommend the next artist. A completely different approach is described in [7], where playlists are generated throughout a 4-step Case-Based Reasoning problem. Finally, Chedrawy et al. [31] used Collaborative Filtering to build playlists.

## 3.3 Implicit playlist recommender (IPR)

The proposed approach extracts recurrent listening patterns from the user sessions and use them to generate recommendations. These listening patterns will be referred to as *implicit playlists*. Given a user, we can identify two types of listening behaviors: “explore” and “exploit”. The former occurs when the user is searching for new songs, the latter when a user is listening to music already known and appreciated. We discard the exploratory behavior as we assume it adds noise to the system and it is not useful in understanding the user preferences. The following steps have been implemented to keep only the sequences of tracks that the user likes and to generate recommendations:

1. For each user, we extract all his listening sessions and compute the *shrunk Jaccard similarity*  $j$  as in (3.1).
2. We maintain only the user sessions if there exists at least another session with a Jaccard similarity over the threshold  $j\_th \in [0, 1]$ . These sessions form the *implicit playlists* of a user.
3. Using *all* the implicit playlists, we build a track-playlist binary matrix - denoted by  $\mathbf{P}$  - where each element  $(m, n)$  is 1 *iff* track  $m$  belongs to the playlist  $n$ . The training makes an intense use of Apache Spark mapping functions such as *join* and *groupByKey*.
4. Given a user listening session, we define the vector  $p_u$  of size equals to the total number of tracks, where the  $m$ -th element is 1 *iff* the user played the track  $m$

		MSD	Celma 1K	Celma 360K	Yahoo! R1	Yahoo! R2	Yahoo! C15	artofthemix	#nowplaying	MMTD	Music Micro	30Music
	Sources	(a)	(b)		(c)			(d)	(e)	(f)		(b)
Features	Release	2011	2010	2010	n/a	n/a	n/a	2003	live	2013	2012	2015
	Span	n/a	4y	n/a	1m	5y	10y	1m	4.5y	18m	1y	1y
	Users	??	1K	360K	2M	1.8M	-	1M	5M	15K	136K	45K
	Tracks	1M	1M	-	-	136K	625K	218K	764K	134K	71K	4.6M
	Artists	44K	180K	160K	9.4K	100K	-	60K	95K	25K	19K	600K
	Playlists	-	-	-	-	-	-	29K	-	-	-	280K
	Track info	x	-	-	-	x	-	-	x	-	-	x
	Acoustic	x	-	-	-	-	-	-	-	-	-	-
	User info	x	x	x	-	-	-	-	-	-	-	-
Interactions	Play events	-	20M	-	-	-	-	-	63M	1M	600K	31M
	Playtime	-	-	-	-	-	-	-	-	-	-	x
	Sequences	-	-	-	-	-	-	-	-	-	-	2.7M
	Ratings	-	-	-	12M	717M	262M	-	-	-	-	1.7M
	Play count	??	-	17M	-	-	-	-	-	-	-	-

Figure 3.1: Music datasets comparison

in this session. In order to give more importance to the most recently listened tracks, the values of  $p_u$  were discounted exponentially in time with a factor  $w$ . Finally, we multiply  $p_u$  by  $\mathbf{P}$  to obtain a relevance score for each implicit playlist. The final recommendation list is generated by aggregating the implicit playlists by decreasing relevance score.

The shrunk Jaccard similarity is computed as follows:

$$j(S_1, S_2, j\_sh) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2| + j\_sh} \quad (3.1)$$

where  $S_1$  and  $S_2$  are the sets of the tracks of two sessions and  $j\_sh$  is the shrinkage factor used to filter out sessions that are too short.

### 3.4 Experiment Setup

Several challenges in the music domain have been only partially explored due to the scarcity of data available to researchers for experiments. For instance, tasks such as user modeling and playlist recommendation require information about listening events (e.g., user, track, time, duration), explicit information about user preferences (e.g., loved tracks, playlists), and user listening sessions. The work done in creating this dataset has been published in RecSys 2015 as a poster paper [127].

#### 3.4.1 Dataset

##### Related datasets

There exist a number of publicly-available music datasets, used in several music experiments. Figure 3.1 compares our crawled dataset – *30Music* dataset – with the existing publicly available datasets.



Most datasets provide content information (e.g., metadata, tags, acoustic features), but only a few report some user-system interactions (e.g., ratings, play events) useful to profile users and to experiment with personalization tasks.

The Million Song Dataset (MSD) [18] is a public collection well-known for its size. It contains audio features (e.g., pitches, timbre, loudness, as provided by the Echo Nest Analyze API<sup>1</sup>) and textual metadata (e.g., Musicbrainz<sup>2</sup> tags, Echo Nest tags, Last.fm tags) about 1M songs (related to 44K artists).

Celma [29] has published two music datasets collected through the Last.fm API: 1K-user and 360K-user. The smaller one - 1K-user dataset - contains the listening habits (20M play events) of less than 1K users. The bigger one - the 360K-user dataset - collects the information about 360K users, but it does not have any listening data other than the number of times a user has listened to an artist. Data are provided as downloaded from the Last.fm API.

Yahoo! Labs have released several music datasets<sup>3</sup>. For instance, the R1 and the R2 datasets provide ratings on artists and songs, respectively, but not user play events.

Some datasets have been extracted from microblogs, such as the Million Musical Tweets Dataset [57].

Finally, The Art of the Mix Playlist dataset<sup>4</sup>, consists of 29K user-contributed playlists, containing 218K distinct songs for 60K distinct artists. However, there are no user listening events.

#### Dataset creation

The *30Music* dataset has been obtained via the Last.fm public API<sup>5</sup>. Last.fm provides free APIs to track details of user listening sessions. In the case a user has connected his supported player to his Last.fm account, the player “*scrobbles*” the user listening activity, i.e., it transfers the play event to Last.fm that records such user interaction.

It is worth noting that only listening events are recorded, while pause/skip events are not scrobbled from the user player to Last.fm, as well as any playlist or explicit preference defined or expressed in the player. The main way for a user to create a playlist in Last.fm is not through the player with the scrobble plugin, but through the Last.fm website; similarly, the user can express explicit preferences (‘love’) about tracks directly in the website. As a consequence, explicit ratings and playlists stored in Last.fm are not biased by external systems (e.g., the recommendations proposed in the player). Unfortunately, Last.fm policy is to scrobble only tracks played for at least half their duration (or for at least 4 minutes), although many listening events lasting less than 5 seconds have been found in the collected data. To build the *30Music* dataset, we started from a list of 2M Last.fm usernames from the Chris Meller dataset<sup>6</sup>.

Given the list of users, we retrieved their playlists (`User.getPlaylists`) together with the single tracks composing the playlists (`Playlist.fetch`). Starting from users with at least one playlist (about 90K users), we retrieved (`User.getRecentTracks`) the user listening events over a 1-year time window (from Mon, 20 Jan 2014 09:24:19).

<sup>1</sup> <http://the.echonest.com/>

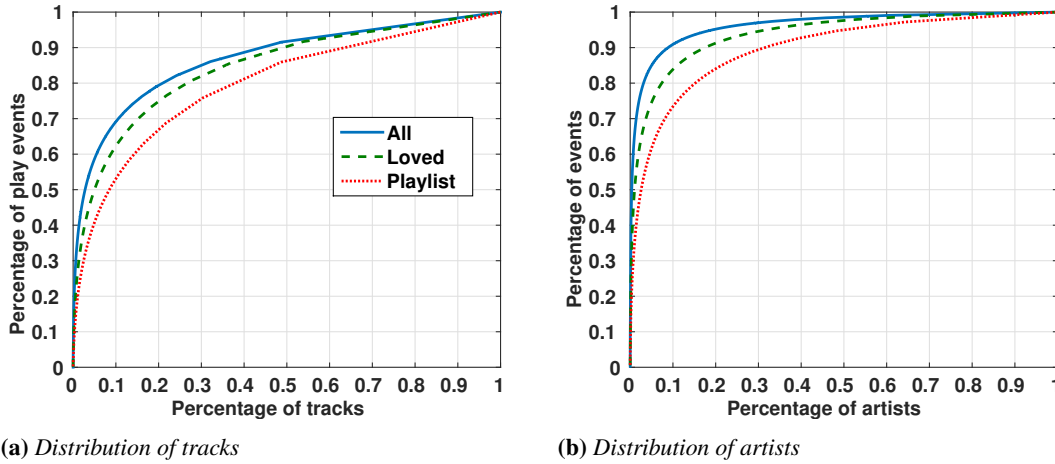
<sup>2</sup> <https://musicbrainz.org/>

<sup>3</sup> <http://webscope.sandbox.yahoo.com/catalog.php>

<sup>4</sup> <http://labrosa.ee.columbia.edu/projects/musicsim/aotm.html>

<sup>5</sup> <http://www.last.fm/api>

<sup>6</sup> <https://opendata.socrata.com/Business/Two-Million-LastFM-User-Profiles/5vvd-truf>



**Figure 3.2:** Cumulative play events distributions for tracks and artists

The raw playlists and user listening events have been enriched with additional information about users (`User.getInfo`) and tracks (`Track.getInfo`).

Furthermore, the data downloaded with the Last.fm API has been processed using Python scripts exploiting some Apache Spark functions for a distributed processing of the massive amount of data. In order to keep only complete and reliable data, we discarded users with some missing data (e.g., if the track scrobbled by the user has the wrong metadata and it is not recognized by Last.fm, the whole user is discarded). In this way, we kept only half of the users, those who have complete information.

Finally, we defined a new entity, the *user play session*, as an ordered list of play events that are assumed to be consequently listened by the user with no interruptions. We define a play event to be part of a session if it occurs no later than 800 seconds after the previous user play event. We determined this threshold by assuming a gaussian distribution of the playtimes frequencies and computing the 99,7% percentile (see Figure 3.6).

**30Music format** The *30Music* dataset is released in accordance with the *Idomaar* data format [115], a multigraph representation oriented to recommender system evaluation that explicitly represents *entities* (i.e., nodes) and *relations* (i.e., edges).

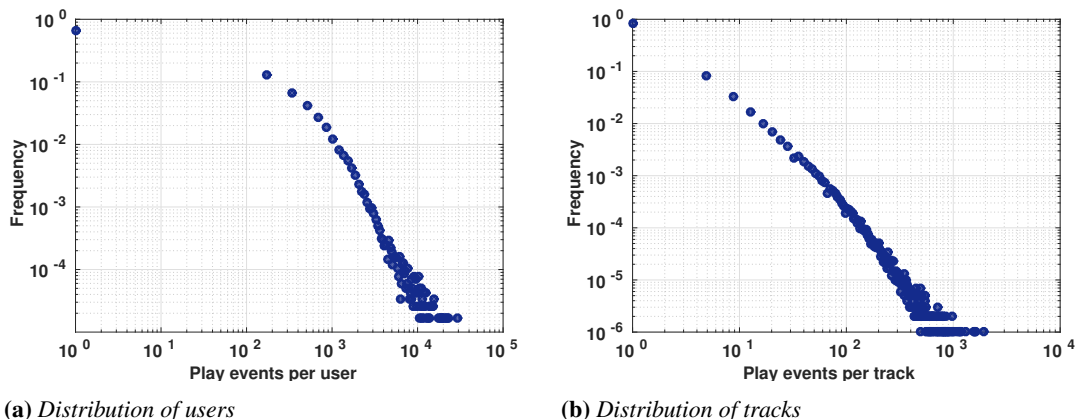
Entities model any object that can be recommended. The dataset is formed by 45K users, 5.6M tracks, 50K playlists, 600K artists, 200K albums, and 280K tags.

Relations model links between two (or more) entities. We defined 31M user play events, 2.7M user play sessions, and 4.1M user love preferences.

### Dataset analysis

The dataset contains 31,351,954 play events organized into 2,764,474 sessions (an average of 11 play events per session). The dataset also contains 1,692,924 explicit ratings (loved tracks), with an average of 33 ratings per user, and 279,351 user-created playlists. The number of events without track duration is 1,277,893 (4.08%).

Figure 3.2a plots the empirical cumulative event distributions as a function of the number of tracks (in percentage). Tracks in the horizontal axis are ordered according



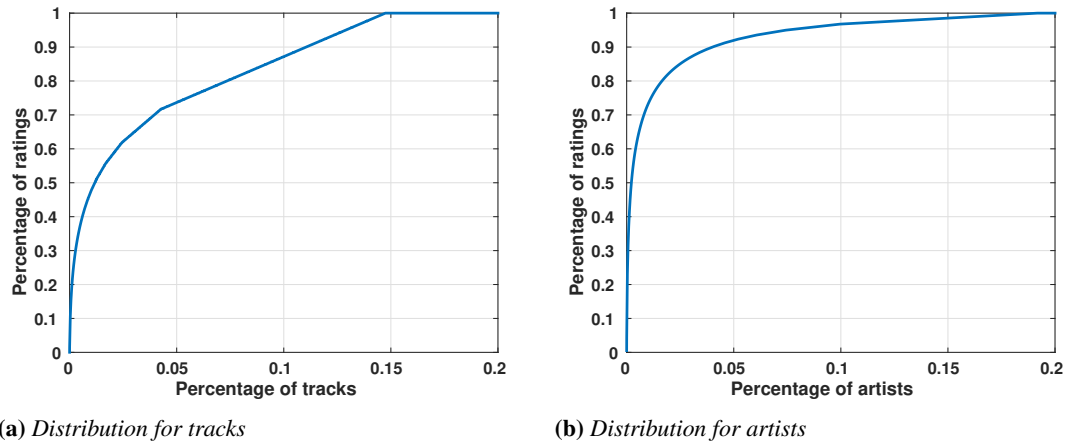
**Figure 3.3:** Power-law distribution of the number of play events per user and track

to their popularity, most popular to the left. The figure plots the same distribution for two subsets of the play events: play events of “loved” tracks (i.e., tracks for which the user expressed an explicit preference) and play events of playlist tracks (i.e., tracks played from a playlist). We observe that play events present a moderate long-tail distribution. The 20% most popular tracks collect 80% of the play events. This long tail effect is mitigated when analyzing preferred tracks (i.e., loved tracks and tracks in the playlists). We observe that the same percentage of play events (80%) involves twice the tracks (40%) when considering tracks in the playlists. We can deduce that users have preferences spanning many different tracks, but their listening behaviour is biased toward the most popular tracks.

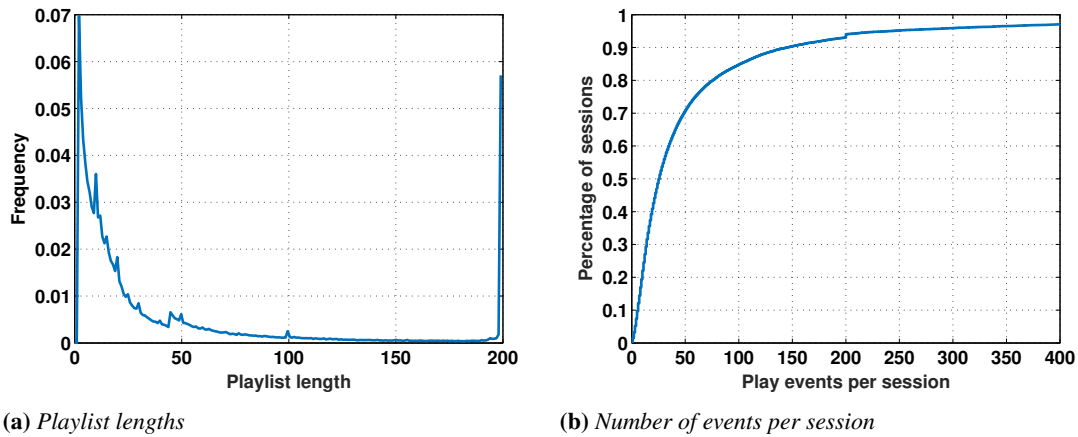
A similar analysis has been performed for artists (see Figure 3.2b). Differently from the previous plot, play events present a strong long-tail distribution: the 20% most popular artists collect more than 95% of the play events. This long tail effect is strongly mitigated when analyzing preferred tracks. We observe that the same percentage of play events (95%) involves 50% of the artists when considering tracks in the playlists. We can deduce that users have preferences spanning many different artists, but their listening behaviour is strongly biased toward the most popular ones.

Figure 3.3 depicts the distribution of the number of ratings per user (3.3a) and per track (3.3b). Both distributions exhibit clear power law characteristics with a *short head* of “popular” tracks and very active users, and a very large set of tracks and users associated with very few events.

Figures 3.4a and 3.4b plot the empirical cumulative explicit rating distribution as a function of the (percentage) number of tracks and artists, respectively. Entities in the horizontal axis are sorted according to their popularity, most popular to the left. Since only 14.73% of the tracks and 19.93% of the artists have received at least one explicit preference, the two plots are limited to the 20% most popular tracks and artists, respectively. We observe that the distribution of the explicit ratings within tracks does not exhibit a strong long-tail behaviour. 5% of the most popular tracks collect 75% of the explicit ratings. On the other hand, the number of explicit ratings is strongly skewed toward a few very popular artists. 5% of the of most popular artists collect more than 90% of the explicit ratings. These results confirm our previous intuitions over users’ listening behaviour. Users tend to love (and to listen to) a few very popular



**Figure 3.4:** Empirical cumulative explicit rating distribution for tracks and artists

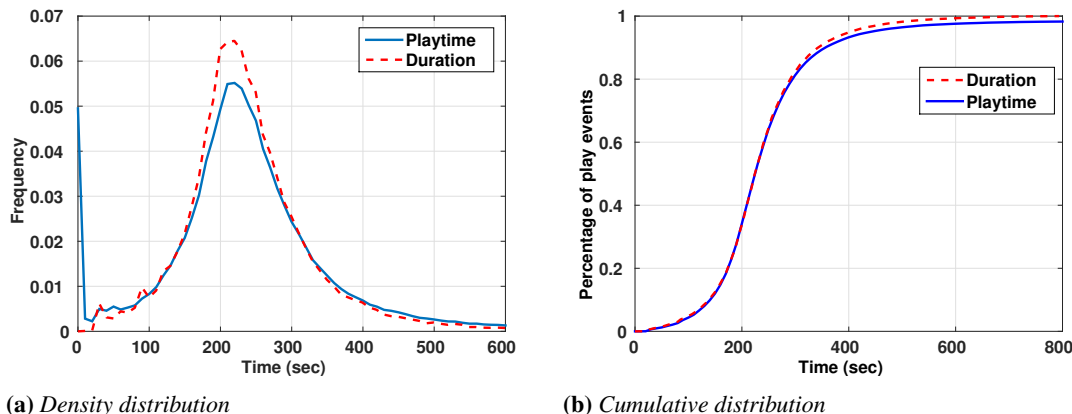


**Figure 3.5:** Probability distribution of the playlist lengths and cumulative distribution of the number of events per session

artists. However, their preference spans across several tracks of these very popular artists. Still, they tend to provide explicit rating for few of the tracks they have listened to. This can be due to the mechanism adopted by Last.fm to collect explicit feedbacks, which forces users to access to the Last.fm online service and to provide their “love” to a track there. This imposes a heavy burden over users, but on the other hand it enhances the strength of each explicit rating, because it is a clear expression of the willingness of the specific user to provide that feedback.

Figure 3.5a plots the percentage of occurrences of playlists as a function of their length. A relatively high number of playlists have only one track. This behavior can be explained as if the user wants to “save” the track for later and can be a proxy of a positive preference of that user for that track. We can also observe that many playlists have length of 200, due to the hard limit posed by Last.fm on playlists’ length.

Figure 3.5b refers to play events in sessions and shows that 70% of the sessions have 50 tracks or less. Note that the small discontinuity for the sessions with a length of 200 events corresponds to users who fully listened to playlists of maximum length.



**Figure 3.6:** Density and cumulative distributions of playtimes and durations of tracks

Figure 3.6 shows the empirical probability density function and the empirical cumulative distribution of the playtime of an event and of the duration of the played song. In Figure 3.6a we can observe a very high value in the frequencies below 5 seconds: these are related to the so called “skipped” songs. Average playtime and average duration coincide: this validates the methodology implemented for generating sessions.

The dataset is made available to the research community<sup>7</sup>. We expect it will foster the exploration of the several challenges still open in the settings of online music applications.

### 3.4.2 Compared Methods

In this section we describe SAGH and CAGH, two algorithms proposed in the literature in the playlist recommendation setting [25] that we used as baselines. Both algorithms suggest popular tracks and, despite of the basic model behind, they proved to outperform more complex solutions such as Collaborative Filtering, Markov models, K-nearest-neighbors, and frequent pattern mining.

**Same Artist - Greatest Hits (SAGH)** This approach [85] considers only tracks of artists that the user has already listened to during the current listening session. The rank of the tracks is then based on their popularity. Thus, when a recommendation is requested, the algorithm searches for the top popular tracks of every listened artist in the current session. The success of this simple algorithm [25] proves the role of artist-awareness and popularity in driving music preferences.

The core of the algorithm consists in computing the top popular tracks (greatest hits) per artist. Using Apache Spark, we sum the play events for each song (*reduceByKey*) and we keep only sessions played more than once (*filter*). Finally, for each artist, we sort the songs in decreasing play event count in order to obtain the artist-track occurrences *map*.

**Collocated Artists - Greatest Hits (CAGH)** This algorithm is an extension of SAGH [25]. Instead of only considering artists just listened in the user session, it also includes the

<sup>7</sup>[http://recsys.deib.polimi.it/?page\\_id=54](http://recsys.deib.polimi.it/?page_id=54)

greatest hits of the most “similar” artists. As a measure of artist similarity, we simply apply cosine similarity on the session-artist matrix, as typically used in collaborative filtering.

### 3.4.3 Evaluation Methodology

We compared the proposed IPR algorithm to the two baseline solutions (CAGH and SAGH) in terms of quality and computing times. We used the user listening sessions available in the *30Music* dataset as input (Section 3.4), splitting them into two groups - training and test sets, in accordance to standard evaluation techniques for recommendation evaluation (e.g., [35]). We divided sessions on a time basis: sessions started before 21 Oct 2015 (9 months data) were included in the training set, the more recent ones (3 months data) in the test set. The training set has been used to bootstrap the recommendation model, either CAGH, SAGH, or IPR. The test set has been used to compute the recommendation quality. For each session  $S^i$  in the test set, we created the known session  $S_k^i$  and the hidden session  $S_h^i$ . The  $S_k^i$  set is composed by the first  $s$  play events of  $S^i$  and it is available to the recommendation algorithm to make all the required inference (e.g., to identify the user session). The  $S_h^i$  set is composed by all play events after the  $S$ -th event. In our experiments we used  $s$  - referred to as *session length* - equals to either 1, 2, 5, 10 or 20. All sessions with a number of play events lower than the required one were excluded from the related test.

For each session, we generate a recommendation list composed by  $N$  tracks, experimenting with  $N = 1, 5, 10, 25$ . Smaller values of  $N$  evaluate the capability of the algorithm to catch the immediate needs of the user, while larger ones better reflect the long-term ones (still within a single user session). The quality has been computed in terms of *recall*, as the percentage of recommended tracks that appear in the  $S_h^i$  set. The algorithm recall is obtained by averaging the recalls of all sessions. We chose the recall since our objective is to retrieve all the tracks that the users actually listened to.

During evaluation it is important to consider that users may listen the same track several times within a session. To evaluate the impact of repetitions on the recommendation quality, we experimented both *with* and *without repetitions* in the test set only. No filtering over repetitions was performed for the training set, leaving the algorithms free to exploit all the information during their training stages.

### 3.4.4 Settings

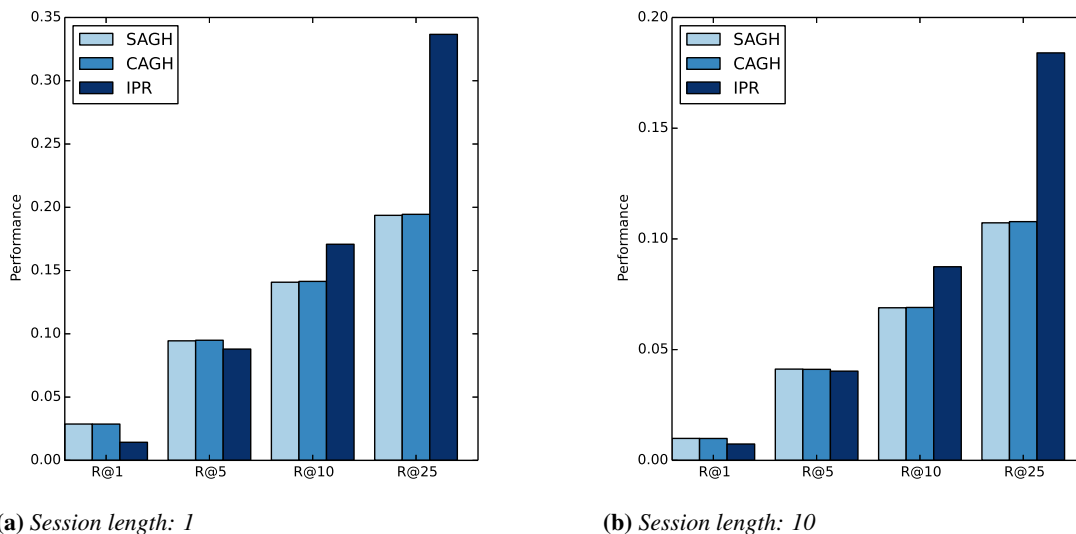
Given the problem size, the computation is particularly demanding and has been implemented using the Apache Spark<sup>8</sup> Python APIs (PySpark) for a scalable and distributed processing.

In order to manage the massive amount of data, we used Idomaar<sup>9</sup> as the evaluator framework, whose core evaluation functionalities (dataset splitting into training, test, and ground-truth sets and quality evaluation) are implemented with distributed and scalable technologies such as Apache Kafka and Apache Spark. The main component of Idomaar is the “orchestrator”, which is designed to manage the incoming streams of data and messages (e.g., user-item interactions/signals, requests for recommendations,

---

<sup>8</sup><https://spark.apache.org/>

<sup>9</sup><http://rf.crowdrec.eu/>



**Figure 3.7:** Recall of the tested methods with repetitions in the test set at different cutoffs of the recommendation list.

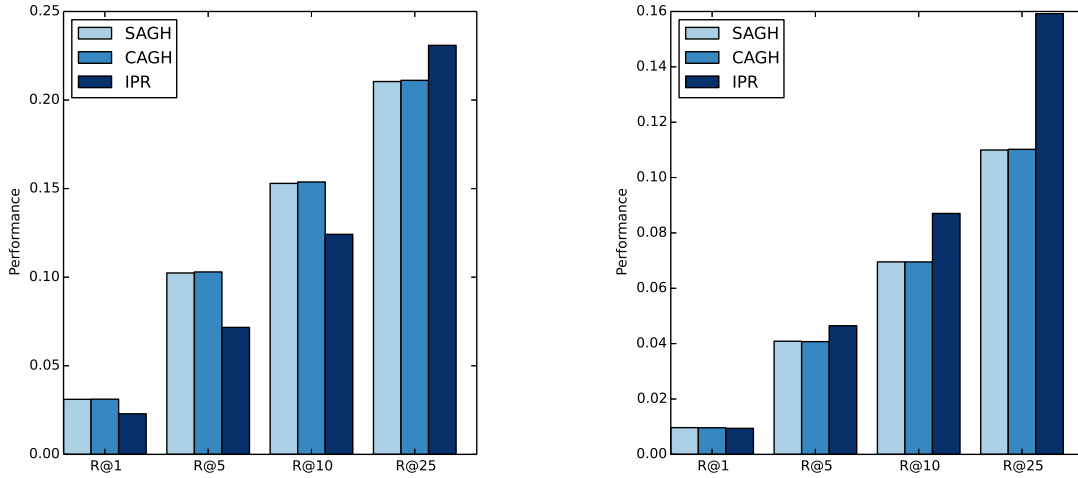
etc.), distribute them to the “computing environment” where the recommendation models are bootstrapped and the recommendation requests are served; finally, the output of the computing environment is sent to the “evaluator” where they are evaluated both in terms of quality and response time.

The experiments have been performed on an Amazon Elastic Compute Cloud (EC2) cluster composed by 9 nodes. Amazon Elastic Compute Cloud is a cloud computing platform which allows users to create, launch, and terminate virtual machines as needed, paying by the hour for active ones. This platform allows to create 3 different types of instances: *on-demand instances* paid by hour without commitment, *reserved instances* with one-time payment, and *spot instances* ran only if the spot price is below the bid specified by bidder. Each node was a *r3.2xlarge* instance equipped with 8 virtual CPUs, 61 GB RAM, and 160 GB SSD, running Apache Spark 1.4.0. One of the nodes was elected to cluster master and was provisioned as on-demand instance, the others worked as cluster slaves and were provisioned as spot instances. Practically, there were 8 nodes dedicated to computation and 1 to orchestrating the process, for a total of 72 virtual CPUs, 540GB RAM, and 1.4TB SSD.

We tuned the algorithms’ hyper-parameters on an independent 10% subset of the users in the dataset. In the end, we configured SAGH and CAGH so that the number of greatest hits per author was equal to 100 and 50, respectively. In addition the author similarity threshold used by CAGH was set to 0.4. As for the IPR, we configured  $j\_sh = 5$ ,  $j\_th = 0.1$ , and  $w = 0.7$ .

### 3.5 Results

Figure 3.7 shows the performance of each algorithm for session lengths 1 and 10, i.e., recommendations are provided to users when they have listened to only 1 and 10 tracks respectively. Repeated tracks were not removed from the test sessions in these experiments. The performance of SAGH and CAGH are almost equal, with a small advantage



(a) Session length: 1

(b) Session length: 10

**Figure 3.8:** Recall of the tested methods without repetitions in the test set at different cutoffs of the recommendation list.

for CAGH for large  $N$  values. IPR always significantly outperforms the baselines for  $N \geq 5$ , with differences ranging from 20% to 70%. With lower values of  $N$ , the baselines still provide better results for small user session lengths ( $s$ ). Figure 3.8 shows the performance of the algorithms when repeated tracks are filtered out from the test set. In this case our algorithm has a smaller increment on the performance. This is due to the fact that our algorithm saves the sessions as implicit playlists also with the repeated tracks, although they are not considered in the similarity function. Hence removing repeated tracks lowers the performance because IPR recommends them in any case. In the extreme case the user session is composed by one track, the baselines clearly outperform IPR. However, when more information is available (i.e.,  $s > 1$ ), IPR is the clear winner for recommendation lists containing at least 5 songs.

IPR suffers from extreme session cold-start situations, i.e., when only few information is available about the context. In these conditions, IPR does not have the necessary support to correctly infer the implicit playlist for the user. Popularity-based algorithms - like SAGH and CAGH - are not affected by this issue and have better performance. However, as the user interacts with the system and provides additional feedback – so that the system can understand its current taste –, the “trivial” popularity criterion fails. In such condition the more sophisticated criterion at the ground of IPR clearly better matches user tastes and provides higher quality recommendations.

Moreover, IPR has been proven to be superior to the baselines for longer recommendation lists. This shows that IPR is able to provide higher quality recommendation with a lower number of recommendation requests with respect to baselines. We believe this is due to the intrinsic capability of IPR to capture track co-consumption patterns, which are highlighted only in longer recommendation lists. SAGH and CAGH are still too biased by popularity for being able to capture such patterns effectively.



### 3.5.1 Computing time

We extracted the computing times in training and generation of the recommendation lists for each tested algorithm by analyzing the execution logs of Apache Spark. No significant differences in computing times were noticed among the different experimental conditions.

SAGH and CAGH run respectively in about 24 and 35 minutes. Their execution times were mainly dominated by the stage in which artists' popularity is counted along sessions. The additional overhead of CAGH with respect to SAGH is due to the quadratic nature of the artist-artist similarity computation.

Surprisingly, IPR generated recommendations in 18 minutes, with a 1.3x and 2x improvement with respect to SAGH and CAGH. We believe these improvements, especially against CAGH, are due to the intrinsic flexibility of IPR, which is able to generate a model through a single pass over the input data and with no need of external metadata, such as artist identifiers.

Finally, it is important to point out that configurations of the tuning parameters, especially the similarity thresholds for CAGH and IPR, may lead to significantly different computing times and response qualities. Algorithm configurations were chosen according to the best speed/quality trade-off we found during the hyper-parameter tuning phase.

## 3.6 Discussion

---

In this chapter we developed IPR, a new context-driven playlist recommendation algorithm, and compared it against two of the best state-of-the-art recommendation algorithms. All algorithms have been efficiently implemented in the Apache Spark programming framework. IPR outperforms the state-of-the-art baselines with longer user sessions. This can be explained by the fact that IPR better recognizes longer and recurrent patterns in user listening behavior. In addition, we improved the running times with respect to the baselines, making it feasible to be deployed in larger production environments.

This chapter confirms that the contextual variable of the session is very important to infer the task and the intent of the user in her listening activity. Our context-driven algorithm is able to address the challenges that break the ImP assumption. The lifecycle of items is handled because as new songs are released and listened, it can update its implicit playlists with new data related to fresh songs. The taste development is not a problem for IPR as it does not build any user profile. Finally the task or intent of the user can be inferred by looking at the current listening session, thus recommending the best playlist according to user *current listening taste*.

The power of our algorithm lies in its flexibility and its capability to instantaneously adapt to the user context, making it feasible to “continue on the same mood” of the user. This creates a new way to listen and discover music, extending the principle of a “seed” song, to a sequence of “seeds” songs, the most recent being the most important.



---

# CHAPTER 4

---

## Jobs

---

In the job recommendation domain personalization is crucial. In this domain the “right” item (job) has to be matched to the “right” user. A good match between a user and an item is found when the user’s skills or ambitions match the job posting description. Nonetheless we will show that context, in this domain, cannot be neglected. The bounded capacity and the ephemeral catalog challenges depicted in Figure 1.1 cause old job postings not to be available anymore. In addition to that, user taste evolves, thus making old interactions not to be representative of the actual user needs in terms of jobs. We show that, for example, prioritizing recent preferences and interaction patterns improves the accuracy of recommendations. This confirms that relaxing the assumptions of the ImP paradigm allows recommender systems to take into account the trendiness of an item and the intent of a user, since more recent preferences are a proxy for the user’s current interest in jobs. Finally we show that by leveraging the feedback of the user in different contexts we are able to improve the performance of Bayesian Personalized Ranking [111], a traditional context-blind collaborative filtering method and its context-aware variant, Bayesian Personalized Ranking with Factorization Machines.

### 4.1 Problem Definition

---

The job recommendation domain presents some unique challenges for recommender systems. First of all, it makes sense to recommend already seen items, since users can browse them many times. Second, jobs have a trendiness and freshness window: it is very likely that old job postings are not available anymore [3]. Third, it is key to this domain to match the appropriate job to the appropriate user as this is not only a matter of personal taste. Each job posting requires a particular set of skills that have to be matched by the user personal skills [15]. Recommending job offerings is really differ-

ent from recommending movies or e-commerce goods, mainly because it is unlikely that a user will watch a movie or buy the very same product again. On job recommendation instead it is very likely that when a user views a job posting once, he will return to that same offering to compare it with other ones, or to apply for an interview.

Recommender systems exploit user interactions in order to generate predictions of user preferences. Today's recommenders are not dependent on users explicitly rating items, but rather are able to exploit implicit feedback. The number of types of implicit feedback, here referred to as *channels*, that can be collected from users is large, and growing. These feedback channels provide positive-only feedback, meaning that they capture preference, but not dispreference. They include view, click, like or follow but also replies, bookmarks, follows, saves, shares, purchases, flags, mentions. The potential posed by these channels raises the question of whether additional forms of feedback add additional value, and, if so, how this value can best be exploited to improve recommender systems. Each feedback channel can be interpreted as a different *contextual signal* of the user's intent, since she may interact with the items in different ways.

In this chapter we tackle the job recommendation problem by two different sides: the first is our approach to The 2016 RecSys Challenge sponsored by XING. XING<sup>1</sup> is a professional social network and a job discovery platform mainly operating in the German area. Our approach uses all the positive feedback in a unique model, without distinguishing between different types of feedback. Although this approach may seem simplistic, we rather focused on the ensemble of various recommendation techniques, which granted our team the fourth place in The 2016 RecSys Challenge (first academic team). In this approach we heavily used time to grasp the current preferences of the user.

The other approach, instead, focuses on improving a single Collaborative Filtering technique, the Bayesian Personalized Ranking, in order to take into account different levels of feedback. We focus on Factorization Machines (FMs) [108], a state-of-the-art factorization method shown to achieve performance on par or beyond that of other models. We choose FMs because they are well suited for the integration of auxiliary data in terms of additional features, and, for this reason, allow us to compare different methods of integrating multi-feedback channels. We combine FMs with Bayesian Personalized Ranking (BPR) [111], a *pairwise* learning-to-rank method. We use BPR because it allows us to learn preferences from unary (positive-only) feedback. We consider channels to constitute different *levels* of feedback, with higher levels reflecting a higher user commitment, and thereby a stronger preference signal, than lower levels. Briefly, the benefit of leveraging channels in BPR is the advantage offered by a better-informed sampling of positive and negative items. BPR learns its model by iteratively optimizing over these pairs. It has been shown that the performance and convergence of the BPR optimization model is largely dependent on how well the pairs of positive and negative items are sampled [111, 110]. Sampling the "right" items results to a faster convergence of the BPR model and makes a contribution to the overall performance [110]. In this section, we move multi-channel feedback to a FM framework to make a direct comparison between exploiting multiple feedback channels directly or via sampling possible.

This approach addresses the following research questions (RQs):

---

<sup>1</sup><https://www.xing.com>

- RQ0: As a sanity check, does a recommender that combines multiple channels of feedback improve its performance over a recommender that uses a single channel only?
- RQ1: Does channel-based, level-informed sampling allow for better exploitation of multiple feedback channels than conventional integration of feedback?
- RQ2: What are the advantages of channel-based level-informed sample (i.e., performance, coverage, time)?
- RQ3: How can we best use levels to inform sampling?

## 4.2 Background

In this section we provide an overview of Factorization Machines (FMs) and Bayesian Personalized Ranking (BPR) to be able to explain pairwise-FM, the model that we use in Section 4.3.2. We also refer to pairwise-FM as BPR-FM, since it uses the BPR optimization criterion.

### Factorization Machines

Factorization Machines are general factorization models that are capable of modeling different factorization approaches by feature engineering [109]. In addition to their superiority with respect to performance and accuracy, FMs offer an advantage because they allow easy integration of additional information in the form of auxiliary features. Factorization Machines have been successfully applied in context-aware recommendation [112], Cross-Domain collaborative filtering [82] and Social Recommendations [78].

The standard FMs are designed for data with explicit feedback. Each user-item interaction is modeled by a feature vector  $\mathbf{x}$  and the corresponding feedback (rating) is specified by a real value  $y$ . The standard FMs model with *two-way* interactions is specified as:

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^n w_j x_j + \sum_{j=1}^n \sum_{j'=j+1}^n w_{j,j'} x_j x_{j'} \quad (4.1)$$

where  $n$  is the number of features,  $k$  is the dimensionality of factorization,  $w_j$  are first order interaction parameters and  $w_{j,j'}$  are second order factorized interaction parameters and are defined as  $w_{j,j'} = \langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle$  where  $\mathbf{v}_j = (v_{j,1}, \dots, v_{j,k})$  is  $k$ -dimensional factorized vector for feature  $j$ .

Model parameters are learned by optimizing a *point-wise* objective function, which is defined over training data  $S$ . The objective function is typically a square loss over training samples plus a regularization term to prevent over-fitting.

In recommendation scenarios involving unary positive feedback (instead of e.g., ratings), the target values  $y$  are not available in terms of real value numbers and a point-wise loss function is not suitable. *Pairwise* models on the other hand, learn to correctly *rank* any pair of items for each user. Previous work has reported that pairwise loss functions are better choices for datasets with unary feedback [111, 93, 118]. In this work, we use a pair-wise loss function for FMs based on the BPR model. Before describing such loss function, we give an overview of BPR in the next subsection.

### Bayesian Personalized Ranking

Bayesian Personalized Ranking (BPR) [111] is a state-of-the-art learning-to-rank method for learning from data with unary feedback. The basic idea of BPR is that for any user  $u$  the items with observed positive feedback are preferred over items without any feedback from  $u$ . With that assumption, BPR creates training data by forming tuples  $(u, i, j)$ , where  $i$  is an item with positive feedback from  $u$  and  $j$  is a negative item (an item without any feedback from  $u$ ).

BPR learns the model parameters  $\Theta$  by maximizing the likelihood function  $p(i \succ_u j)$  for all training triples ( $i \succ_u j$  is read as  $i$  is preferred over  $j$  by user  $u$ ). The likelihood function  $p(i \succ_u j)$  is defined as:

$$p(i \succ_u j) = \sigma(\hat{y}_{uij}(\Theta)) \quad (4.2)$$

where  $\sigma$  is the logistic sigmoid function and  $\hat{y}_{uij}$  is a real-valued utility function that captures the relationship between user  $u$ , item  $i$  and item  $j$ , given model parameters  $\Theta$ . In standard BPR,  $\hat{y}_{uij}$  is defined as the subtraction of individual user-item utility function:

$$\hat{y}_{uij}(\Theta) = \hat{y}_{ui}(\Theta) - \hat{y}_{uj}(\Theta) \quad (4.3)$$

The utility of user-item pairs  $\hat{y}_{ui}(\Theta)$  can be defined by different models such as Matrix Factorization or Nearest Neighbor models [111].

Rendle et al. [111] showed that the maximum likelihood problem in BPR can be reduced to an optimization problem with the following pair-wise objective function:

$$L(\Theta, S) = \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{y}_{uij}) - \lambda_{\Theta} \|\Theta\|^2 \quad (4.4)$$

where  $\lambda_{\Theta}$  are the regularization parameters and  $D_S$  is the set of tuples  $(u, i, j)$ . The BPR optimization problem is solved by Stochastic Gradient Descent (SGD).

Due to the large number of possibilities to create the tuples, the negative items  $j$  are typically sampled from the items set. The choice of sampling method in BPR is crucial on the convergence of SGD algorithm. In [111], it has been suggested that the user-item pairs  $(u, i)$  are sampled from  $S$  uniformly (with replacement) and the negative items  $j$  are uniformly drawn from  $I \setminus I_u^+$ . However, in a later study, Rendle et al. [110] proposed a *dynamic* non-uniform sampling method for negative items where the probability of sampling a negative item is dependent on the current model parameters. They showed that for datasets with tailed item popularity distribution dynamic sampling has a significant influence on the convergence of SGD.

### Pairwise Factorization Machines

Using the BPR optimization criteria, which we described earlier, we can adapt FMs for data with unary feedback. A straightforward approach would be to consider the FMs model in (4.1) as the utility of individual user-item pairs, i.e.,  $\hat{y}_{ui}$ . However, FMs are feature-blind, i.e., for a given feature vector, FMs do not know which features correspond to user, item or auxiliary information. On the other hand, in the BPR algorithm, for a given user-item pair the user should be known to the algorithm in order to sample an appropriate negative item. We therefore introduce a finer-grained representation of FMs where the algorithm is aware of the features.

Let us assume that  $\mathbf{x}_{u,i,\mathbf{z}}$  is a feature-aware representation of an input feature vector in FMs such that  $u$  corresponds to the user,  $i$  corresponds to the item and  $\mathbf{z}$  corresponds to the set of auxiliary features in  $\mathbf{x}$ . The utility of individual interactions  $\mathbf{x}_{u,i,\mathbf{z}}$  can be calculated using equation (4.1). In this case the feature vector  $\mathbf{x}_{u,i,\mathbf{z}}$  can be represented with the sparse form  $\mathbf{x}_{u,i,\mathbf{z}} = \{(u, 1), (i, 1), \{(z, x_z) | z \in \mathbf{z}\}\}$  where  $x_z$  is the value of feature  $z$  and can be either binary or real depending on the type of feature. By replacing  $\mathbf{x}_{u,i,\mathbf{z}}$  in equation (4.1), and expanding  $w_{j,j'}$ , the individual utility function  $f(\mathbf{x}_{u,i,\mathbf{z}})$  can be written as:

$$\begin{aligned} f(\mathbf{x}_{u,i,\mathbf{z}}) = & w_0 + w_u + w_i + \sum_{z \in \mathbf{z}} w_z x_z + \sum_{f=1}^k v_{u,f} v_{i,f} \\ & + \sum_{z \in \mathbf{z}} x_z \sum_{f=1}^k v_{u,f} v_{z,f} + \sum_{z \in \mathbf{z}} x_z \sum_{f=1}^k v_{i,f} v_{z,f} \end{aligned} \quad (4.5)$$

Due to the presence of auxiliary feature  $\mathbf{z}$ , we adapt the definition of likelihood and loss functions in BPR to take into account the auxiliary features. Hence, we define a more general likelihood function for BPR as:

$$p(i \succ_{u,\mathbf{z}} j | \Theta) = \sigma(g_{\mathbf{z}}(u, i, j | \Theta)) \quad (4.6)$$

where  $i \succ_{u,\mathbf{z}} j$  indicates that item  $i$  is preferred over  $j$  by user  $u$  with the presence of auxiliary features  $\mathbf{z}$ . Here  $g_{\mathbf{z}}(u, i, j | \Theta)$  is the utility of pairs of feature vectors in FMs. Similar to (4.3),  $g_{\mathbf{z}}$  can be defined by subtracting the utility functions  $f(\mathbf{x}_{u,i,\mathbf{z}})$  and  $f(\mathbf{x}_{u,j,\mathbf{z}})$ . Thus, by using the definition (4.5),  $g_{\mathbf{z}}$  can be written as:

$$\begin{aligned} g_{\mathbf{z}}(u, i, j | \Theta) = & w_i - w_j + \sum_{f=1}^k v_{u,f} (v_{i,f} - v_{j,f}) \\ & + \sum_{z \in \mathbf{z}} x_z \sum_{f=1}^k v_{z,f} (v_{i,f} - v_{j,f}) \end{aligned} \quad (4.7)$$

Note that the above scoring function is very similar to the Matrix Factorization (MF) scoring function. However, it has an additional term (second line) that takes into account the auxiliary features to score a user-item interaction. The BPR-FM model without auxiliary features would be equal to BPR with MF scoring function. So it is obvious that BPR-FM is a generalization of BPR-MF.

The objective function  $L(\Theta, S)$  can be adapted by replacing  $\hat{y}_{uij}$  with  $g_{\mathbf{z}}$  in equation (4.4). The optimal parameters are found by minimizing this function, i.e.,  $\Theta_{OPT} = \operatorname{argmin}_{\Theta} L(\Theta, S)$ . This optimization problem can be solved by Stochastic Gradient Descent.

### 4.2.1 Related work

Our work shares the goal of using FM on implicit feedback with a few recent studies. Nguyen et al.[93] proposed Gaussian FM, where interaction between feature vectors is modeled with Gaussian kernels. They also proposed a pairwise ranking model where the Gaussian FM model can be used for data with implicit feedback. Qiang et al. [105]

proposed RankFM model that uses an ordinal regression model with FM kernel for micro-blog ranking.

Our proposal is related to other work that strives to improve BPR sampling methods. Gantner et al. [53] proposed a non-uniform sampler for negative items where the probability of sampling negative items are given in advance. Rendle et al. [110] proposed a more advanced sampling method where the probability of sampling negative items are adapted dynamically. Most closely related, is the study done by Lerche et al. [74], which proposed BPR with graded relevance. In this method, sampling is partially done using graded feedback. The authors define weights for grading feedback and feedback with higher weights is preferred to feedback with lower weights. However, their method relies on hand-chosen ratios and the choices of weights depends on some content or temporal information. Our method differs from [74] in the sense that it uses multiple channels simultaneously and does not rely on real-valued grades for implicit feedback.

In more recent work, Pan et al. [99] used an extension of BPR, namely BPR with confidence [131], for learning from *heterogeneous* implicit feedback. Their method exploits heterogeneous feedback by modeling feedback confidence. However, the confidence scores should be given a priori and their method is only tested with explicit movie-ratings.

### 4.3 Multi-Stack Ensemble and Multi-Channel Sampling

---

In this section we present two approaches we used in the job recommendation domain. Our first approach, described in Subsection 4.3.1 consists of an ensemble of many different recommendation algorithms. Ensemble theory states that the more the algorithms are diverse and can learn different relations in the data, the better is the performance of the ensemble [104, 114]. Our second approach, presented in Subsection 4.3.2, propose a different context-driven technique in order to improve the sampling of Bayesian Personalized Ranking algorithm in a multi-channel feedback setting.

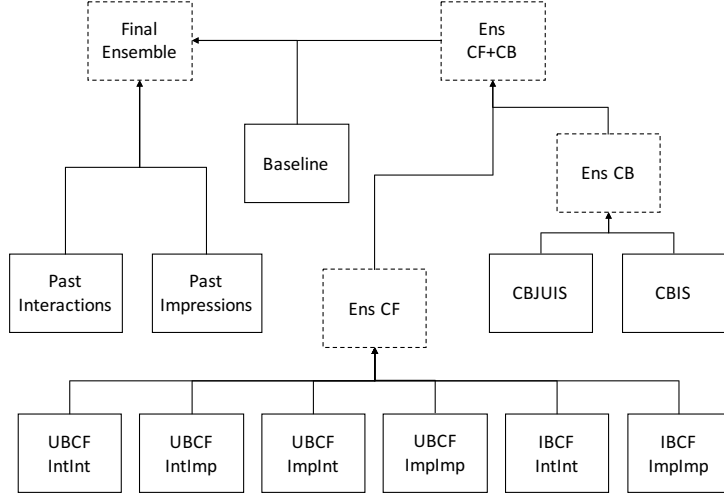
#### 4.3.1 Multi-Stack Ensemble

Figure 4.1 presents an outlook of the layers of our ensemble solution. Algorithms are represented by solid-line rectangles, while ensembles by dashed-line rectangles. The arrows shows the ensemble flow. We used collaborative filtering (CF) algorithms, content-based algorithms, past interactions, past impressions and the baseline. Two ensembling methods were used: linear ensemble and evaluation score ensemble. At the lowest level, we create an ensemble of algorithms exploiting the same technique: for example we merged all the collaborative algorithms and all the content based algorithms, eventually combining them in a single ensemble (CF+CB). For the final submission we created an ensemble merging the past interactions, past impressions, CF+CB and the baseline, thus combining very different techniques and empowering our final ensemble.

#### Algorithms

**Collaborative Filtering** We created a total of 6 collaborative filtering models: 4 User Based Collaborative Filtering (UBCF), and 2 Item Based Collaborative Filtering (IBCF). All of the methods consider every positive interaction as an implicit rating (*signal*) of the





**Figure 4.1:** *Ensemble Stack.*

user’s interest in a particular job. However we converted this implicit signal to an explicit rating as detailed below:

$$r_{ui} = \begin{cases} 0 & \text{if user } u \text{ did not interact with item } i \\ \log\left(\frac{\text{total \# of interactions}}{\text{\# of interactions with job } i}\right) & \text{otherwise} \end{cases} \quad (4.8)$$

In this way the model does not consider every interaction as equally important. The importance of any interaction is higher if the job is less popular. This is inspired by the inverse document frequency (IDF) principle: an interaction with a very common job does has a lower information content with respect to an interaction with a not-so-popular job. We also tried different rating functions (even the binary one), but this was the one which lead to the best performing algorithms. The IDF term is part of a well established technique for normalizing the weight of words in a document, known as term frequency - inverse document frequency (TF-IDF). The term frequency is proportional to the number of times a word appear in a document. We used this principle in the design of many of the algorithms composing our ensemble.

**User Based Collaborative Filtering** Once the rating function is established, we can calculate the user-based similarity. In order to reduce the computational effort and to remove some noise from the datasets, we compute the similarity only between users that have at least a number of rated items above a fixed threshold  $z$ . We define the set  $\mathcal{U}_z$  as the set of all users with at least  $z$  ratings. Given two users  $u$  and  $w$  in  $\mathcal{U}_z$  we compute the user similarity as:

$$sim\_user(u, w) = \frac{\sum_i r_{ui} r_{wi}}{\sqrt{\sum_i r_{ui}^2} \sqrt{\sum_i r_{wi}^2} + \beta} \quad (4.9)$$

where the summations over  $i$  are calculated over the set of all jobs that belong to the samples (impressions or interaction) on which we were computing the similarity. The

**Table 4.1:** *User-Based and Item-based Collaborative Filtering model parameters.*

Name	Similarity	Target	$z$	$\beta$	$K$
UBCF IntInt	Interaction	Interaction	6	7	500
UBCF IntImp	Interaction	Impression	6	7	500
UBCF ImpInt	Impression	Interaction	15	9	500
UBCF ImpImp	Impression	Impression	15	9	500
IBCF IntInt	Interaction	Interaction	15	7	500
IBCF ImpImp	Impression	Impression	10	7	500

shrinkage factor  $\beta$  is used in order to give more importance to users that have a high number of co-rated items.

Having both impressions and interactions data we are able to create two different similarity measures between users, one interaction-based and one impression-based. The final recommendation for user  $u$  and item  $i$  is then calculated as the weighted average of the ratings given to item  $i$  of the top- $K$  similar users to user  $u$ .

With two similarity measures (Int/Imp) and two different options for creating the user profile (Int/Imp), we are able to obtain four sufficiently different predictions for each user that could be mixed together to obtain a better recommendation. Table 4.1 reports the best parameters for each model.

**Item Based Collaborative Filtering** As for the user-based approach, we define the set  $\mathcal{I}_z$  the set of items having at least  $z$  ratings. The similarity between two items  $i$  and  $j$  in  $\mathcal{I}_z$  is calculated as:

$$sim\_item(i, j) = \frac{\sum_{u \in \mathcal{U}} r_{ui} r_{uj}}{\sqrt{\sum_{u \in \mathcal{U}} r_{ui}^2} \sqrt{\sum_{u \in \mathcal{U}} r_{uj}^2} + \beta} \quad (4.10)$$

where  $\beta$  is the shrinkage factor. The predicted rating for an user  $u$  over an item  $i$  can then be estimated as the average of the ratings given by user  $u$  over the top- $K$  similar items to  $i$  weighted by their similarity.

Differently from the user-based approach, the cross combinations IntImp and ImpInt did not yielded good results, so we kept only the IntInt and the ImpImp variants. Table 4.1 reports the best parameters of the used models.

**Content Based Algorithms** We have implemented two different content based algorithms, exploiting the similarity of the concepts of the items (CBIS) and the similarity of the concepts of users and items (CBJUIS).

**Concept-Based Item Similarity** An item is described by a vector in a vector space model composed by its concepts present in the titles and tags of the job posting. We decided not to use other parameters like career level or region because in our analysis we could not find any strong correlation between the user's feature and the job's feature the user interacted with, for example we found top managers interacting with internships. We think this behaviour may be due to the fact that top managers are managers of small start-ups or they want to see how other companies hire people. Each element

### 4.3. Multi-Stack Ensemble and Multi-Channel Sampling

of the vector represents the importance of the feature that we calculated as the IDF [4], thus:

$$w_{ci} = \begin{cases} 0 & \text{if item } i \text{ does not have concept } c \\ \log\left(\frac{|C_I|}{\# \text{ of items having concept } c}\right) & \text{otherwise} \end{cases} \quad (4.11)$$

The similarity between two items  $i$  and  $j$  is then simply defined as the shrunk cosine similarity between the two items vector:

$$sim\_item(i, j) = \frac{\sum_{c \in C} w_{ci} w_{cj}}{\sqrt{\sum_{c \in C} w_{ci}^2} \sqrt{\sum_{c \in C} w_{cj}^2 + \beta}} \quad (4.12)$$

where  $\beta$  is the shrinkage factor. Differently from the CF, for the prediction we used the non weighted average of the similarities of the top 30 similar items to the target item. We set the parameter  $\beta$  to 9.

**Concept-Based Joint User-Item Similarity** We use the concept-based vector space model to represent both users and items in terms of their concepts. A user (document), will be represented by a vector UF of concepts that appeared in the jobs they interacted with in the past. Each element of the vector corresponds to the weight (or importance) of that specific concept for the current user. To calculate these vectors we used TF-IDF normalized weights, calculated as :

$$UF(c, u) = TF_U(c, u) \cdot IDF_U(c) \quad (4.13)$$

$$TF_U(c, u) = \frac{\sum_i b_{uic}}{\sum_i b_{ui}} \quad (4.14)$$

$$IDF_U(c) = \log\left(\frac{\# \text{ users } \in \mathcal{B}}{\# \text{ users } \in \mathcal{B} \text{ having concept } c}\right) \quad (4.15)$$

where  $b_{uic}$  is 1 if the user  $u$  interacted with item  $i$  having concept  $c$  and 0 otherwise and  $b_{ui}$  is 1 if the user  $u$  interacted with item  $i$  and 0 otherwise.

A similar formula was used for the item-feature vector representation. In this case the item represents the document, and the feature represents the term. The difference is that the term-frequency part is binary.

$$IF(c, i) = TF_I(c, i) \cdot IDF_I(c) \quad (4.16)$$

$$TF_I(c, i) = \begin{cases} 1 & \text{if item } i \text{ has concept } c \\ 0 & \text{otherwise} \end{cases} \quad (4.17)$$

$$IDF_I(c) = \log\left(\frac{|\mathcal{I}|}{\# \text{ of items having concept } c}\right) \quad (4.18)$$

Once we have built the vectors for both users and items, we can proceed with calculating the similarity between each pair of user-item. In order to calculate the similarity between user  $u$  and item  $i$  we used the cosine between the angles formed by their vectors  $UF_u$  and  $IF_i$ . Once the similarities for each user-item pair are computed, we sort these similarities for each user  $u$  in decreasing order and recommend the first 30 items.

**Past Interactions** Differently from other domains, for job recommendation it makes sense to recommend already seen items, since users are likely to re-interact with past items for example to review the job description. In order to take into account this information, we processed the past interaction with a filtering and an ordering step. We filtered out all the items whose latest interaction did not happened during the last 2 weeks of activity of each users. We also tried different time windows, however the last two weeks provided the best trade-off between performance and the lowest number of items, leaving more “space” for the other algorithms in the ensemble. Moreover we also did not take into account the “remove” interactions, since they do not constitute neither a positive nor a negative feedback. This is in line with the bounded capacity, ephemeral catalog and taste development challenges depicted in Figure 1.1, which make recent preferences the most representative for the user. We then sorted all the items per descending interaction count, following the principle that the more times an item had been interacted with the more interesting it is for the user, since he had continuously got back to that job posting, thus adapting to the trendiness of the jobs.

**Past Impressions** Impressions, i.e. the output of the XING recommender, cannot be ignored, since they heavily bias the available choices of each user. This simple algorithm takes the last week of each user for which the data is available. Again, this confirms our hypothesis regarding the preference shift, so the last recommendations are the most useful for the user. We noticed that simply reordering these items in the list of recommendation led to very different results with a difference up to 50k points. We tried many reordering methods, but the best performing one was to sort the items for each user using the Concept based joint user-item similarity (See Section 4.3.1).

**Baseline** This algorithm was provided by XING and slightly tweaked by us. For each item we compute a score, giving points for each item feature matching a user feature (like region, discipline, industry). This algorithm lead to poor results, it was used mainly to fill those recommendations that our algorithms were not able to provide.

#### Ensembling methods

We chose to create an ensemble of the previous algorithms using a multi-stack ensemble technique [120], where we created a hierarchy of hybrid models in order to exploit the best out of each single learner we had. Each ensemble in the stack uses a voting-based method combined with a reduce function in order to build the hybrid recommendation, as detailed below:

$$s_{awi} = f_E(\text{rank}_{awi}, \Theta_E) \quad (4.19)$$

$$s_{Eui} = \sum_{a \in E} s_{awi} \quad (4.20)$$

$$\text{rank}_{Eui} = \text{sort}(s_{Eui}) \quad (4.21)$$

where  $a \in E$  is an algorithm in the ensemble  $E$ ,  $s_{awi}$  is the score assigned to item  $i$  of user  $u$  for algorithm  $a$  by the scoring function  $f_E$  dependent on the parameters of the ensemble  $\Theta_E$ .  $s_{Eui}$  is the final score for each item  $i$  of the user  $u$  for the ensemble

### 4.3. Multi-Stack Ensemble and Multi-Channel Sampling

**Table 4.2:** Parameters and public leaderboard scores  $l_a$  for each algorithm and ensemble.

Ens. Name	Algorithm	Ens. Type	$w_a$	$d_a$	$l_a$
CF	UBCF IntInt	Eval	.0339	NA	108k
	UBCF IntImp	Eval	.0322	NA	103k
	UBCF ImpInt	Eval	.0238	NA	91k
	UBCF ImpImp	Eval	.0408	NA	156k
	IBCF ImpImp	Eval	.0314	NA	121k
	IBCF IntInt	Eval	.0300	NA	95k
CB	CBJUIS	Lin	1	.001	128k
	CBIS	Lin	1	.0015	102k
CF+	Ens CF	Lin	2	.001	180k
CB	Ens CB	Lin	2	.0015	140k
Final	Ens CF+CB	Lin	3.98	.001	250k
	P. Interactions	Lin	4	.001	230k
	P. Impressions	Lin	4	.001	400k
	Baseline	Lin	.1	.0001	46k
	Final				622k

$E$ .  $\text{rank}_{Eui}$  is the final rank of each item  $i$  for the user  $u$  calculated by the sort function that sorts the items in descending  $s_{Eui}$  order and takes the top 30 items. If an algorithm provides less than 30 recommendations, the remaining part of the list is filled with lower priority algorithms.

Our idea is that if an item is recommended by more than one different technique it has higher probability to be a good recommendation.

Hereafter, we will describe the different voting techniques that we implemented in order to assign a score to each element of the algorithms inside the stack. Table 4.2 reports the name of the algorithm produced by the ensemble together with the used parameters.

**Linear Ensemble** In the linear ensemble we used two per-algorithm parameters: the weight  $w_a$  of the algorithm  $a$  and the decay  $d_a$  of algorithm  $a$ . The score  $s_{aui}$  is calculated as:

$$s_{aui} = w_a - \text{rank}_{aui} \cdot d_a \quad (4.22)$$

We used integers for  $w_a$  in order to establish a priority over the algorithms, while  $d_a$  has a very low value (in the order of magnitude of  $10^{-3}$ ), so that it does not change the ordering between algorithms with the same  $w_a$ .  $d_a$  is then used as an *interleaving factor* that allows to interleave the recommendations of algorithms with the same priority (same weight  $w_a$ ). We set  $d_a$  proportionally to the public leaderboard score of algorithm  $a$ . This ensemble method is used for Ens Cont, Ens CF+CB and for the Final Ensemble.

**Evaluation Score Ensemble** The Evaluation Score ensemble assigns to each item in the recommendation list a score that reflects the maximum score that can be obtained using the provided evaluation metric by recommending a relevant item in that particular position.

$$s_{aui} = w_a \cdot e(\text{rank}_{aui}) \quad (4.23)$$

where  $e(\text{rank}_{aui})$  is defined as:

$$e(\text{rank}_{aui}) = \begin{cases} 37.83, & \text{rank}_{aui} \in [1, 2] \\ 27.83, & \text{rank}_{aui} \in [3, 4] \\ 22.83, & \text{rank}_{aui} \in [5, 6] \\ 21.17, & \text{rank}_{aui} \in [7, 20] \\ 20.67, & \text{rank}_{aui} \in [21, N] \end{cases} \quad (4.24)$$

For determining the weights  $w_a$  we defined the *score density ratio* as the ratio between the leaderboard score and the total number of recommended items:

$$w_a = \frac{l_a}{n_a} \quad (4.25)$$

where  $n_a$  is the number of items recommended by algorithm  $a$  and  $l_a$  is the public leaderboard score for algorithm  $a$ . We use this ensemble method for the Ens CF.

### 4.3.2 Multi-Channel BPR-FM

In this subsection, we first introduce the way channel information can be embedded into the BPR-FM model and then we propose a multi-channel sampling method for BPR-FM.

#### Multiple Channels as Auxilliary Features

The most straightforward way to exploit multiple types of positive feedback in BPR-FM is to use the types (*channels* or *contexts*) of the feedback as auxiliary features in the FM model. The basic assumption here is that the type of feedback contains some information that reflects the commitment level or preference of the user for the item. With the BPR-FM model, the type of feedback can be considered as an additional discrete feature. Figure 4.2 illustrates the matrix of a pairwise FM with three positive (white background) and three negative (red background) samples. In this example, items are playlists and the types of feedback are ‘click’, ‘share’ and ‘like’. The three types of feedback are considered as discrete features. For each positive example, an artificial negative example is created in such a way that the user and auxiliary features (in this case ‘feedback channel’) are the same as the positive example and the item is a sampled item from  $I \setminus I_u^+$  (note that here  $\mathbf{x}_1^+$  is an example of  $\mathbf{x}_{u,i,z}$  and  $\mathbf{x}_1^-$  is an example  $\mathbf{x}_{u,j,z}$  according to the BPR-FM model).

The advantage of such representation is that the types of feedback are not scored explicitly and the strength of the feedback can be learned by the algorithm. The negative examples can be sampled similarly to the standard sampling method of BPR or by the multi-channel sampler that we introduce in the next section.

#### Channel-informed Level-based Sampling

This section presents our approach, which improves BPR-FM by using level-based sampling exploiting multiple feedback channels. In [80], we introduced the idea of multi-channel sampling for BPR. Here, we provide a fully developed version of the initial idea by integrating multi-channel sampling to BPR into FM, decoupling the sampling model for positive and negative items, and carrying out extensive testing.











$x_1^+$	1	0	0	1	0	0	0	1	0	0
$x_1^-$	1	0	0	0	0	1	0	1	0	0
$x_2^+$	0	1	0	0	1	0	0	0	0	1
$x_2^-$	0	1	0	0	0	1	0	0	0	1
$x_3^+$	0	0	1	0	0	0	1	0	1	0
$x_3^-$	0	0	1	1	0	0	0	0	1	0
										
	Users			Items				Type of feedback		

Figure 4.2: Embedding feedback channels in Pairwise Factorization Machines.

Assume that  $p(u, i, j)$  is the probability distribution from which tuples  $(u, i, j)$  are sampled. This distribution can be expanded as:

$$p(u, i, j) = \underbrace{p(u, i)}_{\text{positive pair sampler}} \cdot \underbrace{p(j|u)}_{\text{negative item sampler}} \quad (4.26)$$

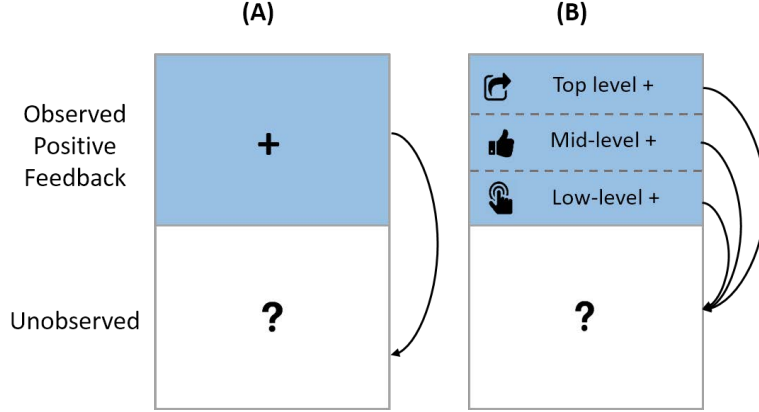
The two components in the above probability distribution are *positive pair sampler*, by which user-item pair  $(u, i)$  is sampled, and the *negative item sampler*, by which  $j$  is sampled.

In standard BPR,  $(u, i)$  is selected by uniform sampling from the training set<sup>2</sup>  $S$ , and  $j$  is sampled uniformly from  $I \setminus I_u^+$ . Thus, the probability distribution of the two samplers in the standard BPR can be denoted by  $\frac{1}{|S|}$  for positive pairs and  $\frac{1}{|I \setminus I_u^+|}$  for negative items.

For multi-channel positive feedback we introduce the notion of a feedback *level* that reflects the importance of a feedback channel. The notion of level is also used to define an ordinal scale for different feedback channels. The higher the level of a feedback is, the higher is the user's interest or commitment level to the item. For example, in a music recommendation framework, a user 'like' should have a higher level than a 'click', assuming that a 'like' is a stronger indication of interest than a 'click'. Figure 4.3 illustrates the standard versus the multi-channel sampling for the BPR-FM model. The arrows show the preferences. The assumption of multi-channel sampling is that an item at a higher feedback level is preferred to an item at a lower feedback level.

Our multi-channel level-based sampler differs from the standard BPR sampling method in two ways: First, the multi-channel method samples additional relations compared to the standard BPR sampling. In fact, the positive and negative items in tuples  $(u, i, j)$  can also read as a 'preferred' and a 'less preferred' item. Second, the probability of sampling positive and negative item pairs depends on the level in which they occur.

<sup>2</sup>Note that the user-item pairs can also be selected by iterating over training data but it has been shown that sampling with replacement (bootstrapping) is more efficient than iteration [111].



**Figure 4.3:** Standard (A) versus multi-channel sampling (B) for the BPR-FM model. The arrows indicate preferences.

Our multi-channel sampler utilizes feedback channels to over- or under-sample certain feedback types to better learn the user’s preference model. Thus, the probability of sampling positive and negative item pairs also depends on the level where they appear.

We denote  $\mathbb{L} = (L_1, \dots, L_p)$  as the ordered set of levels in a dataset such that  $L_i \succ L_{i+1}$ , that is, feedback in  $L_i$  are stronger signals of interest compared to the feedback in  $L_{i+1}$ . For example, in Figure 4.3 there are four levels: three positive and one unobserved. We also denote  $\mathbb{L}^+$  as the set of positive levels (since the feedback level can be also negative, such as dislike). Let  $L$  and  $N$  denote the level of the positive and negative item in tuple  $(u, i, j)$ . The multi-feedback sampler samples tuples  $(u, i, j)$  from the following set:

$$D_{MF} = \{(u, i, j) | i \in I_{L,u} \wedge j \in I_{N,u} \wedge L \in \mathbb{L}^+ \wedge L \succ N\} \quad (4.27)$$

such that  $I_{L,u}$  represents the set of items in levels  $L$  that user  $u$  interacted with and  $I_{N,u}$  represents the set of items that can be sampled in level  $N$  as negative item.

As already stated, in the multi-channel feedback sampling method the probability of sampling the tuples depends on the levels. We can therefore assume that the tuples are sampled from a joint probability distribution  $p(u, i, j, L, N)$ , which can be expanded as:

$$p(u, i, j, L, N) = p(u, i, L) \cdot p(j, N | u, L) \quad (4.28)$$

In the above equation, the positive pair sampler and the negative item sampler are both joint probability distributions with positive or negative levels. We now define different distributions for positive and negative samplers.

**Positive Pair Sampler** In Multi-channel sampling, the positive feedback sampler can be further expanded as:

$$p(u, i, L) = p(u, i | L) p(L) \quad (4.29)$$

where  $p(L)$  is the probability of sampling a positive level. We define  $p(L)$  as following:

$$p(L) = \frac{w_L |S_L|}{\sum_{Q \in \mathbb{L}^+} w_Q |S_Q|} \quad (4.30)$$



### 4.3. Multi-Stack Ensemble and Multi-Channel Sampling

where  $|S_L|$  is the number of feedback signals in level  $L$  and  $W_L$  is a weight associated to a level that should reflect the importance of the level. The weight parameters can be either static values or can be dynamic values depending on model parameters. In our preliminary experiments we found that dynamic weights typically cause the model to be overfitted. We found that the reciprocal rank of the levels are good candidates for the weights of the levels. Given a level, the positive user-item pair is then sampled uniformly from that level. To summarize, the positive pair sampler first samples a level according to (4.30) and then a user-item pair is sampled uniformly from the sampled level. With the above sampling method, the positive feedback from higher levels has higher chance to be sampled. This chance is determined by the size and weight of levels.

**Negative Item Sampler** Similar to (4.29), the negative item sampler can also be expanded to finer-grain distributions:

$$p(j, N|u, L) = p(j|u, N)p(N|u, L) \quad (4.31)$$

where the first probability distribution is used to sample a negative item given a negative level, and the second distribution is used to sample a negative level given the user and the positive level.

In [80], using *only* the unobserved level as the negative level typically results in a better model. We therefore assume that  $N$  is always the unobserved level and thus we can simplify the negative sampler as  $p(j|u)$ . We propose three different distributions for the negative item sampler:

**Uniform-Item:** Here the negative item is sampled uniformly from the negative level and by considering the unobserved level as negative level,  $p(j|u)$  can be defined in a similar way as sampling negative item in the standard BPR:

$$p(j|u) = \frac{1}{|I \setminus I_u^+|} \quad (4.32)$$

**Uniform-Feedback:** With this sampling method, first a feedback (a  $(u', j)$  pair) is sampled from the set of all positive feedback that are provided by other users and then  $u'$  is discarded and  $j$  is considered as the negative item. This is equal to popularity over-sampling since the popular items have higher chance to be sampled. Here  $p(j|u, L)$  can be defined as:

$$p(j|u) = \frac{|(u', j') \in S : u' \neq u \wedge j' = j|}{|S|} \quad (4.33)$$

**Multi-Channel:** With this method both the popularity of an item and the level of its feedback are taken into account for sampling it as a negative item. Here a  $(u', j)$  pair is sampled from the joint distribution  $p(u', j, L'|u)$ , similar to (4.29), and then  $u'$  is discarded. The probability distribution of this sampler can be defined as:

$$p(j|u) = \begin{cases} p(u', j, L'|u) & j \in I \setminus I_u^+ \\ 0 & \text{otherwise} \end{cases} \quad (4.34)$$

The intuition behind this sampler is that the popular and more important items, if not rated by the target user, are better candidates as negative items. Rendle et al. [110]

```

1: procedure LEARN BPR-FM( $S, p_{pos}, p_{neg}$ )
2:   initialize  $\Theta$ 
3:   repeat
4:     sample  $\mathbf{x}_{u,i,z}$  from  $S$  according to  $p_{pos}$ 
5:     sample  $j$  according to  $p_{neg}$  and create  $\mathbf{x}_{u,j,z}$ 
6:     let  $g_z(u, i, j|\Theta) = f(\mathbf{x}_{u,i,z}|\Theta) - f(\mathbf{x}_{u,j,z}|\Theta)$ 
7:     update  $\Theta$  according to BPR update rule [111]
8:   until convergence
9:   return  $\Theta$ 
10: end procedure

```

**Figure 4.4:** Learning BPR-FM with Stochastic Gradient Descent and adapted samplers.

showed that SGD typically converges faster if the the model is less certain about the correct order of the sampled items. By considering the more *important* (higher level and more popular) items as negative, the uncertainty of BPR for finding the true rank between the pairs can increase and thus the model can converge faster.

All the above probability distributions can be pre-computed in advance and thus the sampling can be done in  $\mathcal{O}(1)$ . Therefore the computational complexity of the multi-channel method is the same as the standard BPR model. Figure 4.4 lists the BPR-FM learning algorithm with adapted positive pair and negative item samplers.  $p_{pos}$  refers to the positive pair sampler which can be (4.29) or can be a uniform distribution.  $p_{neg}$  is negative item sampler and can be either (4.32), (4.33), (4.34) or other possible sampling methods.

## 4.4 Experiment Setup

---

We carried out our experiments using the XING dataset, used for The 2016 RecSys Challenge hosted by XING. In the first experiment we used the evaluation provided by XING, while in the second we used traditional metrics, such as Mean Reciprocal Rank.

### 4.4.1 XING Dataset

XING is a professional social network and a job discovery platform. The XING dataset has been used for The 2016 RecSys Challenge<sup>3</sup>. In this dataset, the items are job postings and user positive feedback is collected through three different channels: `click`, `bookmark` and `apply`.

The dataset provided is a subset of the interactions of the XING users in a time span of 12 weeks. The (hidden) test set is relative to the positive interactions (`click`, `bookmark`, `reply`) of the test users during the week immediately following the training set. The task is to compute 30 recommendations for each test user among the recommendable (active) items.

The data provided for the challenge consisted of:

$\mathcal{U}$ : 1.5M records of users and their features.

$\mathcal{I}$ : 1.3M records on items and their features.

$\mathcal{A}$ : 10.1M records on users and the job offerings that the XING recommender system showed the user during the training set period, grouped by week. Xing does not ensure

---

<sup>3</sup>2016.recsyschallenge.com

that these were actually seen by the user.

$\mathcal{B}$ : 8.8M records on users and their interactions with job offerings over the training period. For each user-job pair also the type of interaction (click, bookmark, reply to offer, hide from view) and their timestamps were present.

$\mathcal{C} = \mathcal{C}_U \cup \mathcal{C}_I$ : The set of the concepts ( $\sim 100k$ ) relative either to the users ( $\mathcal{C}_U$ ) or to the items ( $\mathcal{C}_I$ ). Some of the records in set  $\mathcal{U}$  and  $\mathcal{I}$  have missing features.

Both  $\mathcal{U}$  and  $\mathcal{I}$  share some features, such as the industry id, the discipline id, the region id, and the career level. The users have also the feature relative to the concepts of their career, while the items have the same concepts extracted from the title and the description of the job posting.

#### 4.4.2 Experiment 1: The 2016 RecSys Challenge

##### Evaluation methodology

The test set consists of 150k users that were all present in  $\mathcal{U}$ , but not all of them were also present in  $\mathcal{A}$  or in  $\mathcal{B}$  or in both: 107.4K of them were in both  $\mathcal{A}$  and  $\mathcal{B}$ , 25.2K of them were in  $\mathcal{A}$  and not in  $\mathcal{B}$ , 2.7K of them were in  $\mathcal{B}$  and not in  $\mathcal{A}$ , and 14.7k users were the cold start users, i.e. they did not have any interaction or impression. Among the 1.3M job offerings, only about 300k were recommendable (active) during the test period.

The task of the challenge is to provide, for each user, an ordered list of at most 30 recommended items. The evaluation metric for the challenge combines different metrics like *precision at k*, *recall* and *user success*.

The evaluation metric for the challenge combines different metrics like *precision at k* ( $p@k$ ), *recall* and *user success*:

$$p_u@k = \frac{reclist_u@k \cap groundTruth_u}{k} \quad (4.35)$$

$$recall_u = \frac{reclist_u \cap groundTruth_u}{|groundTruth_u|} \quad (4.36)$$

$$userSuccess_u = \begin{cases} 1 & \text{if } reclist_u \cap groundTruth_u \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \quad (4.37)$$

where  $reclist@k_u$  is the recommendation list for user  $u$  of length  $k$  (with no subscript it is the entire recommendation list),  $groundTruth_u$  are the items in the test set for user  $u$ ,  $p_u@k$  is the precision at  $k$  for user  $u$ ,  $recall_u$  is the recall for user  $u$ , and  $userSuccess_u$  measures if there is at least an item in both  $reclist_u$  and  $groundTruth_u$ .

The score for user  $u$  is then calculated as follows:

$$s_u = 20(p_u@2 + p_u@4 + recall_u + userSuccess_u) + 10(p_u@6 + p_u@20) \quad (4.38)$$

and the final score is the sum of the scores for all test users.

We point to [1] for more details about the dataset, the evaluation and the general challenge organization.

### 4.4.3 Experiment 2: Multi-Channel BPR

#### Evaluation Methodology

The performance of recommendation are measured with different ranking metrics including Precision, Recall, MRR (Mean Reciprocal Rank) and NDCG (Normalized Discounted Cumulative Gain). Due to high correlation between the above metrics we only report the results based on MRR and Recall with a cutoff of 10. To calculate the metrics we used an evaluation technique known as *one-plus-random* [35, 16].

All the experiments have been done with four-fold cross-validation in order to limit problems like overfitting and to give an insight on how the model will generalize. The ground truth is compliant with XING needs, so we use any positive feedback. In order to reduce data sparsity and the size of dataset, we filtered out all the users and jobs that overall have less than 20 feedback in all channels.

The experiments are implemented within an open source toolkit [81]. The number of latent factors (parameter  $k$ ) is set to 10. The latent factors are initialized by a normal distribution with a mean of 0 and standard deviation of 0.1. The SGD algorithm is used with the following hyper-parameters. Learning-rate: 0.05, regularization parameter: 0.002. As mentioned, the XING (RecSys Challenge 2016) dataset is publicly available, making it possible to replicate the experiments exactly.

## 4.5 Results

---

### 4.5.1 Experiment 1: The 2016 RecSys Challenge

Table 4.2 shows the public leaderboard score for every single algorithm and ensemble of our method. The ordering of past impressions and interactions already provides good results, therefore from every other algorithm we excluded the items that were present either in the past interaction or in the past impressions for each user. In this way the recommendations would have been different in each of our methods, thus giving more diverse information to the ensembling algorithm. This is the reason for the relatively low performance of collaborative and content algorithms: they do not contain the items that the user is more likely to interact with. Without any filtering their scores are much higher.

We can observe that any ensembling method is effectively exploiting the differences in the algorithms, always delivering better performance. Taking Ens CF as an example, it scores 180k compared to the best performance of the best CF algorithm, i.e. 156k. The Eval ensembling method was used only in the Ens CF: this can be explained by the fact that CF algorithms' confidence in recommendations degrades quickly and a linear ensemble is not able to weight them in the proper way. The linear ensembling, instead, has been proven the best for the other ensembles: by controlling the weight and the decay it basically produces an interleaving with priority: a higher weighted algorithm is recommended before a lower priority one, while maintaining its internal ordering. Two equally weighted algorithms, instead, are interleaved proportionally to their decay ratio: this is the case of the Final Ensemble, where interactions and impressions have the same decay and weight, so the final ensemble will interleave an item from the past impressions and an item from the past interactions. Of course, if an item is present in more than one algorithm contributing to the ensemble, this is pushed higher in the

**Table 4.3:** Performance of single-channel versus multi-channel training methods.

Channels	MRR@10	Recall@10
Single Channel: Reply	0.0386	0.0162
Single Channel: Bookmark	0.0233	0.0159
Single Channel: Click	0.2647	0.1608
Combined: Oracle experiment	0.2835	0.1690
Combined: Standard BPR	0.2471	0.1501
Combined: BPR+channels	0.2582	0.1519
Combined: MCF-sampling	0.3034	0.2010

recommendation list.

## 4.5.2 Experiment 2: Multi-Feedback BPR

### Exploiting Multiple Channels

We first carry out experiments to confirm that multiple channels of feedback do indeed improve the performance of a recommender (RQ0). This sanity check is necessary to have explicit evidence that multiple channels contain different information, and that adding additional channels does not duplicate information. Table 4.3 reports experimental results on the XING data. The “Single Channel” rows are standard FMs trained with the feedback from one channel only, the “Combined” rows report on approaches exploiting all channels.

First, we discuss **Combined: Oracle Experiment**, which represents an upper bound on any late fusion approach that we could develop. The oracle chooses the best channel for each prediction and uses that channel. The results show that the oracle improves over all of the individual channels. This provides an answer to our RQ0: We now can be assured that additional channels have the potential to provide additional information.

Next, we investigate what happens when we use *all* feedback information available, but do not differentiate between channels. All feedback information is considered equally important. We do this by conflating all channels to a single “generic interaction” and using BPR-FM. **Combined: Standard BPR**, which employs BPR-FM. This method does not outperform our oracle experiment, which motivates us to continue investigation of approaches which do not conflate levels.

The most straightforward way of exploiting levels is to add channel information as different categories of auxiliary information in the BPR-FM model (see Section 4.3.2 and Figure 4.2). This approach is labeled **Combined: BPR+channels** in Table 4.3. It shows that adding channel information as auxiliary features outperforms Standard BPR. However, ultimately, this use of channel information is disappointing since it does not outperform the oracle experiment, meaning that it would be possible to train a late fusion approach that could outperform it in a given application scenario.

Finally, we turn to approaches that differentiate levels using sampling. We focus here on *Multi-Channel-Full* sampling, which is labeled here as **Combined: MCF-sampling**, and is investigated later in more detail. This method differentiates levels for both positive and negative sampling. We see that this method beats both the Oracle (representing the best possible late fusion approach) and Standard BPR (representing the best possible approach that does not differentiate levels). This experiment allows

us to answer RQ1 positively: level-informed sampling does indeed allow for better exploitation of multiple feedback channels. This fact clearly demonstrates the advantages of context-driven sampling methods, i.e. Combined: MCF-sampling, over context-aware methods, i.e. BPR+channels, confirming that the simple addition of the context in personalized methods has room for improvements.

### Using Channels for Sampling

Next, we turn to methods that exploit channels using channel-based level-informed sampling.

There are two approaches to choose positive samples: uniform and multi-channel (Eq. (4.30)), and four approaches to choose negative samples: Uniform-Item (Eq. (4.32)), Uniform-Feedback (Eq. (4.33)), Popularity-Oversampling (Eq. (4.33)) and Dynamic-Oversampling ([110]). These sampling options creates eight different combinations for sampling. The MRR@10 achieved by these approaches is illustrated in the top plot of Fig. 4.5.

Training time is shown in the second plot of Fig. 4.5. It is reported in epoch time, which is the average time of one iteration on the training data. The time complexity of the Multi-Channel samplers is in the same range as uniform samplers. The higher time cost of Dynamic-Oversampling methods can be attributed to the fact that each update has an additional complexity of  $\mathcal{O}(k)$ .

Coverage is shown in the third plot of Fig. 4.5. It is reported the total percentage of items that is recommended to all users. Here the approaches involving Dynamic-Oversampling and Uniform-Item sampling fall short.

We see that the Multi-Channel positive sampler always outperforms the Uniform-Feedback positive sampler with any negative sampler either in MRR@10 and in coverage, achieving the lowest epoch time. As for the negative sampler, the results of Uniform-Feedback and Multi-Channel are very close, but the former achieves more item coverage.

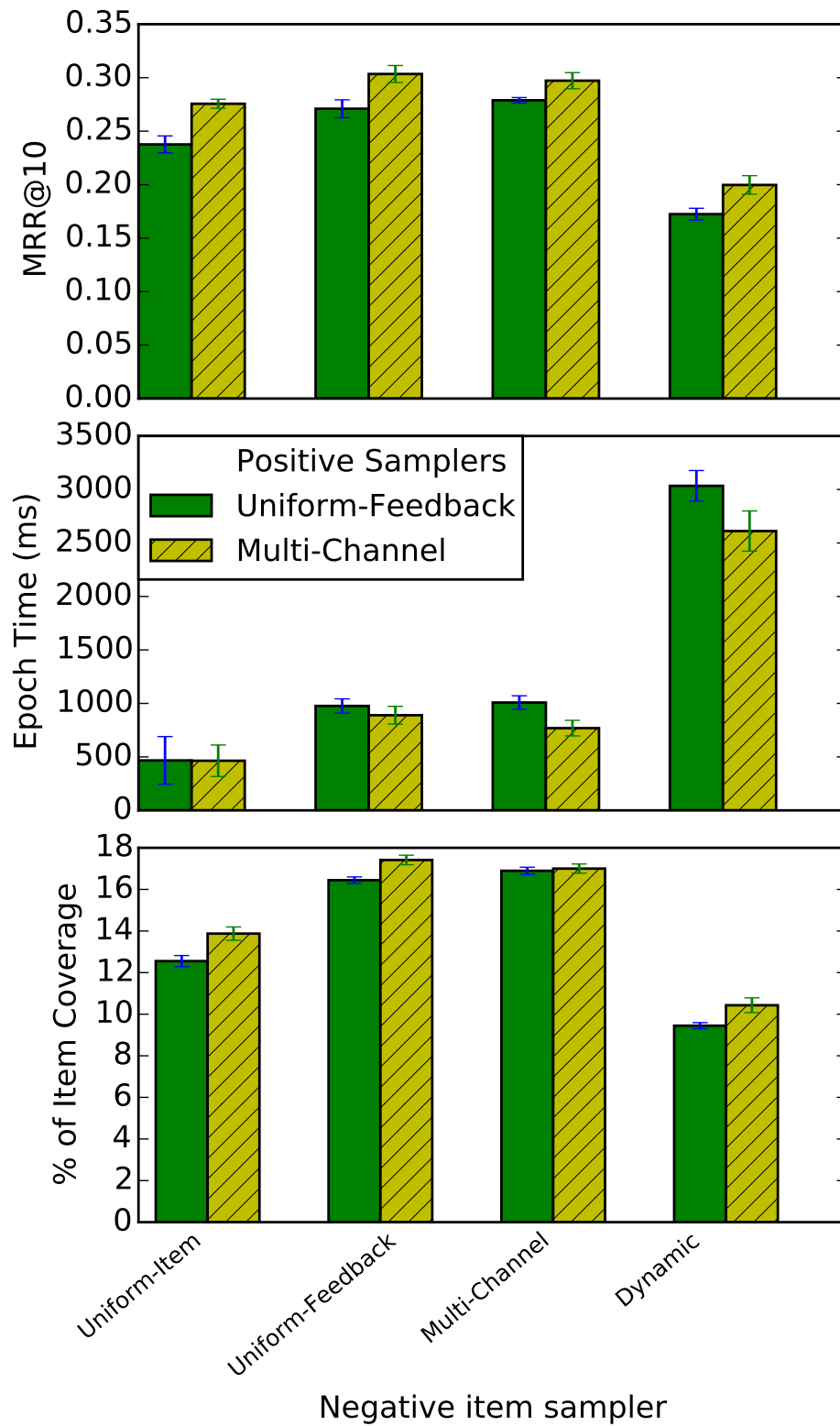
To summarize, we have found that channel-based, i.e. context-driven, level-informed sampling provides advantages in time complexity and in coverage, improving prediction performance, which we turn to next.

### Multi-channel Samplers

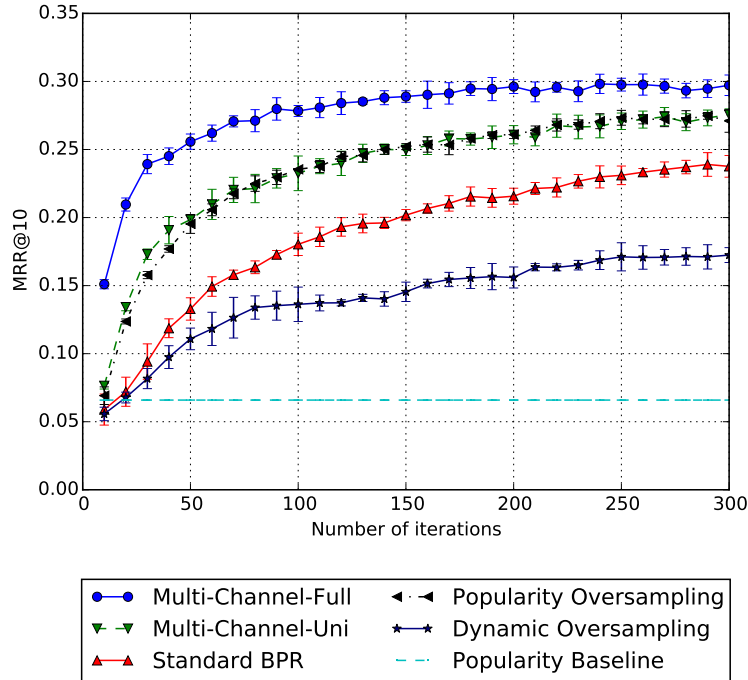
We turn to detailed experiments on channel-based, level-informed sampling methods for the considered dataset. The following sampling strategies were chosen as most interesting and informative: **Multi-Channel-Full**: In this case the positive user-item pairs and negative items are both sampled informed by level. The positive pair and the negative item is sampled according to Eq. (4.29) and Eq. (4.34).

**Multi-Channel-Uni**: With this sampling method the positive user-item pairs are sampled with the multi-channel method, i.e., Eq. (4.29), but the negative items are sampled uniformly according to the sampling distribution (4.32). With this method the channel information is only exploited for sampling the positive pairs.

**Standard BPR**: This case is the same as the standard BPR sampling method. The positive pairs are sampled uniformly from dataset and the negative items are sampled uniformly from the items set.



**Figure 4.5:** Comparison of different sampling method for the BPR-FM model (300 iterations).



**Figure 4.6:** Comparison of different sampling method for the BPR-FM model.

**Popularity-Oversampling:** With this method the popular items are oversampled as negative according to Eq. (4.33). The positive pair sampler is the same as the standard BPR.

**Dynamic-Oversampling:** This is the sampling method that has been introduced in [110]. The probability of an item to be sampled as negative is dynamically updated during the training phase according to the current model parameters. Positive user-item pairs are sampled uniformly the same way as the standard BPR.

The BPR-FM model is trained using the SGD learning method with 300 iterations and after every 10 iterations the model is evaluated. Figure 4.6 shows the performance of the five sampling methods after an increasing number of iterations. Each point in the curve is the average MRR@10 of the four-fold cross validation experiment. The error bars are the standard deviations of the four folds. The five sampling methods are also compared with the most-popular baseline.

Consideration of the graphs in Figure 4.6 shows that the channel-based approach outperforms standard BPR. This observation confirms again our answer to RQ1 given in Section 4.5.2. Exploiting levels during sampling improves over conventional forms of integrating feedback. Specifically, the improvement over BPR is improvement over a sampling method that does not exploit channels, and so, contexts.

## 4.6 Discussion

As for the first experiment, XING claims that its algorithm can score much better on the same data of the challenge. However, the evaluation introduces a bias that underestimates the proposed algorithms. The introduced bias is relative to the impressions: of course it is very likely that a user clicks on items that are recommended by the XING



recommender, as this is the main way for users to interact with the system. This fact imposes a bias on the ground truth, because it is very likely to be a subset of XING recommendations: we believe that evaluating the algorithms proposed by different teams in an online setting would improve the reported performances, since the users are presented with the actual recommendations. The way in which the evaluation has been carried out, instead, promotes the algorithms that learn which are the best items among the ones recommended by the XING platform. Thus any comparison with the XING recommender is penalized. As reported in Table 4.2, the algorithms recommending only the last 2 weeks of interactions and the last week of recommendation have very high scores, confirming that using recent preferences to infer the user intent and situation has a significant impact on the recommendation performance.

The second experiment uses BPR as a learning-to-rank approach within a Factorization Machine Framework in order to fully explore the contribution that multiple channels of user feedback can make to recommendations. We show that multiple feedback channels are useful, and that context-driven, level-based sampling outperforms the more straightforward, but relatively naive, approaches of late fusion or of integrating channel information as auxiliary features, i.e. context-aware techniques. Multi-channel samplers are shown to consistently outperform standard BPR sampling. Our future work will explore further models such as list-wise learning-to-rank methods for use with multi-channel positive feedback. In this experiment, we assume that the order of the levels are known in advance. Further work can be done on also ‘learning’ the right order of the levels.

In this chapter we showed that in a highly personalization and skill driven domain, i.e. job recommendation, the exploitation of context leads to achieve better performance. In the work of The 2016 RecSys Challenge, described in Section 4.3.1, we demonstrated that time plays a fundamental role in defining the short term preferences of job seekers. From these short term preferences the concepts of interest of the user can be derived by the concepts of the items the user has interacted with, thus making the algorithm infer the intent of the user. For the second experiment we conducted an extensive research on how a context-driven level-informed sampling can outperform well established personalization-only and context-aware methods. This chapter confirms that context is not marginal even in domains that do not clearly exhibit contextual behaviors. We showed that hybridization of personalization and context-driven algorithms is not only possible, but can significantly improve the performance of state-of-the-art recommendation algorithms.



---

# CHAPTER 5

---

## Linear TV Program Recommendation

---

Linear TV is a domain that poses hard challenges to the recommendation algorithms, depicted in Figure 1.1. First of all new programs are frequently added to the catalog and it is rare that an already aired program will come back in the future. This makes it impossible to adopt a collaborative approach, since they require the presence of past feedback. Moreover, this domain presents other challenges to the recommender systems community: the TV consumption process is seasonal with different periods: year, season, month, week and day. Each item has a life-cycle: it is consumed very much at the beginning, with the interest of the viewers declining with time. Moreover users' tastes usually change with time because they develop. All these challenges can be addressed by targeting the recommendations to the context, instead of to the user and the item. Context in this domain has a great potential: the company of the user, the time of the day, the day of the week, the channel and the set of programs aired in a specific moment are contextual variables that, intuitively, can affect the recommendation. We found out that the time and channel contextual variables are enough to provide high quality recommendations. This chapter tackles the Linear TV program recommendation problem by heavily leveraging contextual information and describes the work [36] and [125].

### 5.1 Problem Definition

---

Television is one of the most popular media in our era, and with the advent of digital TV and the growing offer of satellite services there are at any time of the day hundreds of available TV programs to be watched by the users on hundreds of different channels. On one hand the user is satisfied by this abundance since the vast choice of programs supports her tastes, but on the other hand she suffers an information overload problem.

This information overload makes the user prone to a tedious channel surfing in order to find what he/she really likes, inevitably leading to annoyance.

In the past the solution was represented by channel guides on paper that used to be consulted on a daily basis. Nowadays, these paper supports are fallen into disuse due to the proliferation of channels and shows and the advent of smart TVs and smart devices. The show schedule information has been embedded into the television software itself through the so-called *electronic program guide (EPG)*. However, the low quality of the EPG in terms of content and its often crude user interface brings to a poor user experience and, as a natural consequence, to ineffectiveness. The answer to this problem consists in providing the user with a short list of recommended programs, representing the subset of the on-air ones that most correspond to her preferences.

While recommender systems have seen tremendous achievements in the field of Video on Demand (VoD), limited effort has been conducted to build an effective recommender system for linear TV. Compared to conventional recommender systems, recommending TV programs is more challenging for several aspects, introduced in Figure 1.1:

- **Ephemeral catalog.** In traditional domains, such as VoD, the available content is updated at a relatively slow rate (e.g., on a daily-basis a few movies are added to/removed from the catalog). Conversely, since TV programs are scheduled at specific times, each item is available only at specific time intervals, leading to a catalog of items that constantly changes over time. This accentuates the *new-item problem*, as many upcoming TV programs have never been watched in the past. Therefore, a TV recommender system cannot rely purely on Collaborative Filtering (CF) techniques to recommended these programs.
- **Time-constrained catalog.** In contrast to VoD which provides users with the ability to select and view content at their convenience, in traditional linear TV programs are broadcast according to a preset schedule, and recommendations should consider only programs transmitted at the moment of the recommendation, or within few minutes [96]. We refer to these programs as *live programs*.
- **Seasonality.** Many TV programs exhibit seasonality effects: some programs are aired only a specific day per week (e.g. TV series), others are aired every day at a certain time slot (e.g. news). Others are aired in a particular moment of the year (e.g. World Cup, special events).
- **Strong context-aware consumption patterns.** TV viewers have a number of preferred TV channels (e.g., the first 9 channels on the remote control) and they prefer to watch TV during specific time periods (e.g., prime time) [61]. In linear TV recommender systems, context can be more important than items. If there is an interesting-enough program for the user in her preferred channels and time slots, the user will watch it, regardless of the fact that a more interesting program could be scheduled on a different channel or at a different time slot.
- **A user cannot watch different TV channels simultaneously.** Unlike traditional recommender systems, where items are always available and a user can consume more items at the same time, in linear TV a users cannot watch more than one program at the same time and many of the not-watched programs will not be aired

again in the near-future. Therefore, when analyzing historical viewing habits, a recommender system should consider that some TV programs which are mutually exclusive because they can be scheduled simultaneously [96].

- **Users might watch the same TV program multiple times**, either the same identical program (e.g., an interesting movie the user likes to watch again) or different “episodes” of the same TV series.
- **The user feedback is usually implicit**, provided in the form of watched/not watched shows.

Note that the first issue makes it impossible to adopt traditional collaborative filtering (CF) recommendation techniques. Indeed, they are not able to recommend new items since such items cannot be compared with the other ones in terms of the feedback provided by the users in the past [28].

Moreover, a fundamental aspect to be considered in TV program recommendation is the *context* [45], i.e. the situation that the user is experiencing when watching television. The most important *contextual signals* for the TV domain are the time and the social setting in which the user is accessing the content, and her current interest topic. So, for instance, when alone during daytime the user might prefer different shows with respect to those liked with friends in the evening.

Although this domain presents also the taste development property, this was not taken into account due to the short time span of the dataset. The life-cycle property, instead, can be associated with the time dependency of the TV program: a program is very much viewed in the beginning (e.g. prime time) and its viewers decrease with time (as night approaches).

### 5.1.1 Notation

We denote by  $\mathcal{C}$  the set of TV channels, by  $\mathcal{A}$  the attributes of a TV program (genre and sub-genre), by  $\mathcal{I}$  the set of TV programs and by  $\mathcal{X}$  additional contextual variables related to the view event. Additionally, we denote by  $\mathcal{F}$  the set of all possible features as the union of channels, attributes, contextual variables and program ids:

$$\mathcal{F} = \mathcal{C} \cup \mathcal{A} \cup \mathcal{X} \cup \mathcal{I}$$

We define the *television rating tensor*  $r_{ufs}(t)$  as the total number of minutes user  $u$  spent watching programs with feature  $f$  in time slot  $s$  during day  $t$ .

For instance, let us assume that on day  $t$  user  $u$  watches 20 minutes of the TV program “Mickey Mouse” (feature  $f_1$ ) of genre “Children” (feature  $f_2$ ), sub-genre “Cartoon” (feature  $f_3$ ) on “Disney Channel” (feature  $f_4$ ) with his family (feature  $f_5$ ) spanning slots  $s_1$  (e.g., Sunday, 2.00PM–3.00PM) and  $s_2$  (e.g., the same day, 3.00PM–4.00PM). This event increments by 20 minutes the 10 elements of the user-preference tensor corresponding to the five features  $f_1 \dots f_5$ , two slots  $s_1$  and  $s_2$  and program  $i_1$ . More formally, for each  $f \in \{f_1, f_2, f_3, f_4, f_5\}$  and  $s \in \{s_1, s_2\}$  we have that  $r_{ufs}(t) \leftarrow r_{ufs}(t) + 20$ .

Elements of  $r_{ufs}(t)$  can be easily computed from the raw data (by joining EPG and tuning events) and can be used to estimate how much a user prefers to watch TV programs of specific genres, on specific channels, or during specific time slots. The

user-preference tensor resembles the user-item-context tensor of traditional context-aware recommender systems [2].

Note that although time is a contextual variable, due to its peculiarity and for simplicity of notation, we represent it as separate from other contextual variables.

### 5.2 Background

---

Recommendation of TV programs has attracted some interest in the recent literature. The existing proposals can be divided on the basis of their aim: recommending to build a personalized video recorder (PVR), or recommending to build a personalized EPG in linear television.

A personalized video recorder is a system generating recommendations about TV content that will be stored into an internal hard disk, for a possible future viewing by the user. The work of Engelbert et al. [50] characterizes TV programs with attributes extracted from an EPG, containing information about channel, title, subtitle, genre, actors, year and description. Recommendations of programs to be recorded are generated on the basis of an initial user profile and an adaptive user profile, both sets of TV programs classified as liked or disliked. The initial user profile is manually filled by the users, while the adaptive one is built using implicit and explicit feedback collected after the user has watched the programs. Once defined the users' profiles, the attributes of new programs (taken from the EPG) are compared against those of the programs in the user's profile with the help of a bayesian classifier. Another personalized video recorder is defined by Kurapati et al. [72]; they too propose algorithms for PVRs coupling explicit and implicit feedback, in this case relying on neural networks to combine them. Srinivas et al. in [147] present an algorithm for PVRs which produces recommendations on the basis of both implicit (using both Bayesian classifier and Decision Trees) and explicit user information, combined together by means of neural networks. The problem analyzed in this work is related to ours but is not the same, because in PVRs the recommendations do not have to be provided at specific time instants.

One of the first paper on the personalized EPG direction is [44]. A number of other works focus on the concept of personalized EPG. Cotter and al. in [121, 32] present a personalized EPG where the selected TV shows are based on a hybrid recommender system that mixes collaborative and content-based recommendations. Users manually input their preferences about channels, genres, and viewing times. This information is combined with the user's viewing activity by means of case-based reasoning and collaborative filtering techniques. Ardissono et al. in [6] present a personalized EPG where recommendations are generated locally on the client side using a hybrid approach on the basis of three information sources: *(i)* user's implicit preferences represented in terms of program categories and channels (content attributes are downloaded from the satellite stream), *(ii)* user classes, and *(iii)* user viewing activity.

Other papers present recommender systems based on hybrid collaborative and content based approaches [5]. The work in [62] describes a TV recommender systems which combines together content-based and collaborative filtering by means of Neural Networks. The system also uses information on the users, such as demographic information, interests, and moods. Martinez et al. in [14] exploit a hybrid approach to solve new-item, cold-start, sparsity, and overspecialization problems. The methods mix to-

gether in the same interface the outcome of content-based (computed using the cosine similarity among item feature vectors) and collaborative filtering (using Singular Value Decomposition to reduce the size of item's neighborhood).

Fewer papers present TV recommender systems based on social networks [55]. Chang et al. in [30] suggest that TV user preferences can be learned from past viewing experience and from friendship connections in social networks. Some works focus their attention on implicit and explicit rating elicitation. Überall et al. in [128] present a recommender system for DVB (Digital Video Broadcasting) based on both the viewing behavior and explicit user preferences on preferred genres, sub-genres, and TV programs. The work in [72] presents a TV recommender system based on a multi-agent approach which combines implicit and explicit user preferences. Implicit preferences are processed with a Bayesian classifier to compute the likelihood that the user will like or dislike a TV program. A decision tree is later used to compute program recommendation scores.

Few works focus their attention on the time evolution of TV recommender systems. Cremonesi et al. in [38] highlight that different CF algorithms have different behaviors according to the state of the recommender system (measured in terms of statistical properties of the user-rating-matrix): during the cold start of the recommender system item-based methods outperform matrix-factorization methods. The work in [96] highlights the problem of when recommending TV shows. The paper suggests the adoption of a push mechanism, i.e., it is not the user who requests a recommendation, but it is the system that, on the basis of a profit model, triggers recommendations at the right time.

Chang et al. [30] provide guidelines to create a TV program recommender, identifying the main needed modules and performance requirements. However, the proposed framework is interesting, but just sketched; among the full-fledged proposals, just a few rely on contextual information.

Some non-contextual linear TV recommenders have appeared in the literature, and many of them rely on hybrid (collaborative and content-based) systems. Barragans-Martinez et al. [14] exploit a hybrid approach to solve new item, cold-start, sparsity and overspecialization problems; their method uses both implicit and explicit feedback, and mixes together the outcome of content-based filtering, computed using the cosine similarity between item feature vectors, and collaborative filtering, exploiting singular value decomposition. Ali et al. [5] develop TiVo, a television-viewing service for the US market incorporating a recommender system which exploits an item-item form of collaborative filtering mixed with bayesian content-based filtering; the system envisages client and server components, and relies on both implicit feedback and explicit ratings. Another hybrid approach is that of Cotter et al. [32], who present a personalized EPG; users manually input their preferences about channels and genres, and this information is combined with the user's viewing activity by means of case-based reasoning and collaborative filtering techniques. Überall et al. [128], on the contrary, propose a fully content-based technique, exploiting both the viewing behavior and explicit user preferences on preferred genres, subgenres and TV programs.

Context is taken into account by Ardissono et al. [6], who develop a content-based system able to generate a personalized program guide. In order to model the user, the system employs several information sources: users' explicit preferences, estimates on

program_id	time	channel	minutes
p1	daytime	Ch-1	20
p2	night	Ch-2	10
p3	daytime	Ch-1	30
p4	night	Ch-1	20

Figure 5.1: Log of Example 5.3.1

$$\begin{matrix} & \text{daytime} & \text{night} \\ \text{Ch-1} & \left( \begin{matrix} 50 & 20 \end{matrix} \right) \\ \text{Ch-2} & \left( \begin{matrix} 0 & 10 \end{matrix} \right) \end{matrix}$$

Figure 5.2: Projection of the tensor of Example 5.3.1 for user  $u$

viewing preferences using program categories and channels, viewing preferences of stereotypical viewers classes, socio-demographic information, and users’ viewing behavior. Different modules of the system manage the different kinds of information, and the results are then combined; the context is considered by the module that estimates user preferences on the basis of the user viewing behavior, since those preferences depend on day and time. Another contextual system is that of Hsu et al. [62]. They propose a hybrid system that combines the collaborative and content-based components by means of a neural network; the contextual information employed by the system is the user’s mood, considered as a strong influencing factor in program selection.

All the described approaches to linear TV recommendations, both the contextual and the non-contextual ones, exploit some form of explicit feedback which must be provided by the users, like, for instance, user ratings. On the contrary, the system we propose relies only on the availability of implicit feedback in terms of history of the past program views, which is the most realistic situation. Moreover, our algorithms exploit contextual information in a more general way with respect to what is done in [6] and [62]. In fact, [6] and [62] deal only with specific kinds of context information, while we devise a framework that can accommodate every type of context dimension, like the kind of people present during the program view or the fact that it is a weekday or the weekend.

### 5.3 Discrete and Smoothed Contextual Recommender

---

This section describes two different TV program recommendation algorithms: the Discrete and the Smoothed Contextual Recommender. The first is simply recommending the programs aired in the most frequent contexts. The second is an extension of the first algorithm, but aggregates the information in contexts which are close to each other.

#### 5.3.1 Discrete Contextual Recommender

**Example 5.3.1.** Consider a set of context dimensions including only the time slot and a set of TV program attributes constituted only by the channel. The possible values for the time context dimension are daytime and night, while those for the channel are Ch-1 and Ch-2. Figure 5.1 shows a possible log of syntonizations for user  $u$ .

In this example the tensor  $T$  has three dimensions: user, time slot and channel. Figure 5.2 shows the projection of the tensor on time and channel for user  $u$ .



### 5.3. Discrete and Smoothed Contextual Recommender

Once the tensor model above has been built, it can be used at runtime to generate the recommendations. The user  $u$  requests recommendations in a given time instant  $t$  when in context  $\mathcal{F}_1 = f_1, \dots, \mathcal{F}_m = f_m$ . Let  $\mathcal{I}_t$  be the set of programs on air at time instant  $t$ . The system extracts from the tensor the appropriate score  $r_{ufs}$  for each item  $i \in \mathcal{I}_t \subset \mathcal{F}$ , as follows:

$$\hat{r}_{ufs} = \sum_t r_{ufs}(t) \quad (5.1)$$

If  $N$  recommendations are required, the system retrieves the  $N$  programs with the highest values of  $\hat{r}_{ufs}$ .

**Example 5.3.2.** Consider the situation described in Example 5.3.1, suppose that the system generates recommendation lists of length 1 and the user  $u$  has requested recommendations at instant  $t$  in the context  $time=night$ . Suppose that at instant  $t$  Ch-1 is showing program  $p5$  while Ch-2 is showing program  $p6$ , therefore  $\mathcal{I}_t = \{p5, p6\}$ . According to the tensor in Figure 5.2, the score for  $p5$  computed using Equation (5.1) is 20 while the score for  $p6$  is 10. Therefore, the system recommends program  $p5$  to  $u$ .

#### 5.3.2 Smoothed Contextual Recommender

In order to better model time, we defined time-slots in a week on a per-hour basis, thus resulting in  $24 * 7 = 168$  finer grain time slots. Moreover we modeled the contextual feature “time” in a smooth way by considering the effect of slots that are close temporarily: e.g. the hour before or after. Since the familiar context in the previous algorithm was not improving the performance for all the users, we decided to remove it from the contextual variables considered.

Our algorithm extends the baselines (i) by introducing the *smoothed time context* and (ii) by incorporating in the model all the features (i.e., genre and sub-genre).

The discrete definition of time context described in the previous section might create unrealistic discontinuities between adjacent time slots. For instance, according to (5.1), a user might like “Cartoons” on “Disney Channel” from 10AM to 11AM but have no interest for “Cartoons” on “Disney Channel” from 11AM to 12AM. This might occur if the viewing history of the user contains many events in the first time slot (10AM–11AM) and fewer events in the second time slot (11AM–12AM).

In order to mitigate this effect, we introduce a smoothing function which aggregates the user preferences in each time slot with the preferences in the neighbor time slots.

We first define a distance function which measures the distance between time slots:

$$d(s_1, s_2) = \left[ \text{days}(s_1, s_2) + \frac{\text{mins}(s_1, s_2)}{L} \right] \sigma(s_1, s_2) \quad (5.2)$$

where

- $\text{days}(s_1, s_2)$  is the difference in days between  $s_1$  and  $s_2$  (e.g., the distance between Tuesday and Thursday is 2, as well as between Tuesday and Sunday)
- $\text{mins}(s_1, s_2)$  is the difference in minutes between the time of day of  $s_1$  and  $s_2$  (e.g., the distance between 00:05 and 00:15 is 10, as well as between 00:05 and 23:55)
- $\sigma(s_1, s_2)$  is 1 if  $s_1$  and  $s_2$  are both weekends or both weekdays, 0 otherwise.

Given a time slot  $s$ , we define its neighbor set  $\mathcal{N}(s)$  as  $\mathcal{N}(s) = \{s' | d(s, s') \leq D\}$ , where  $D$  is the maximum aggregation distance. In our experiment we have used  $D = 6$ . In this way, for instance, the time slot 'Monday 20:00' is in the same neighborhood as 'Friday 20:00' and 'Monday 16:00', but not with 'Friday 16:00'.

Given a time slot  $s$ , we can aggregate its neighbors  $\mathcal{N}(s)$  in several ways. We explored two main functions: (i) by score and (ii) by rank.

**Score aggregation** The *score-aggregated* user-preference tensor  $a_{u f s}$  is computed by smoothly merging the user preferences of time slots in  $\mathcal{N}(s)$  by means of a weighted average:

$$a_{u f s} = \frac{\sum_{t, s' \in \mathcal{N}(s)} r_{u f s'} \cdot \beta(s, s')}{\sum_{t, s' \in \mathcal{N}(s)} \beta(s, s')} \quad (5.3)$$

where  $\beta$  is a weighting function that depends on the distance. In our experiments we have defined  $\beta(s, s')$  as

$$\beta(s, s') = \frac{1}{d(s, s') + 0.1}$$

**Rank aggregation** The aggregated user profile for time slot  $s$  is generated as follows. We sort the time slots  $s' \in \mathcal{N}(s)$  in increasing order of distance from  $s$  and we label them with their rank distance  $k(s')$ , starting with  $k(s) = 0$  for slot  $s$ , the closest to itself. Time slots  $s_1$  and  $s_2$  with the same distance from  $s$  have the same rank value  $k(s_1) = k(s_2)$ .

We now define the *rank-aggregated* user-preference tensor  $a_{u f s}$  as

$$a_{u f s} = \max_{s' \in \mathcal{N}(s)} \left[ \delta_{u f s'} \cdot e^{-\alpha k(s')} \right] \quad (5.4)$$

where  $u$  is the user,  $f$  is the feature,  $s'$  is the slot,  $\delta_{u f s'}$  is 1 if  $\sum_t r_{u f s'}(t) > 0$  and 0 otherwise, while  $\alpha$  is a smoothing constant. This aggregation function assigns the highest score (i.e., 1) to all the features related to TV programs watched by the user during time slot  $s$ . Other features, related to programs watched during more distant slots, have a lower score, which decays exponentially to 0 with increasing rank distance from  $s$ .

In our experiments we have set  $\alpha \approx 0.07$  so that the score of a feature is 50% of the maximum score when  $k = 10$ .

**Recommendations** Let us assume that a TV program  $i$  is scheduled in a time window that spans a set of time slots defined as  $S_i$ , and it is characterized by a set of features – among which the channel – defined as  $F_i$ . The preferences  $\hat{r}_{ui}$  of user  $u$  on live TV program  $i$  can be estimated as

$$\hat{r}_{ui} = \sum_{s \in S_i, f \in F_i} \frac{a_{u f s} \cdot w_f}{w_f \cdot |S_i|} \quad (5.5)$$

where  $|S_i|$  is the number of slots spanned by program  $i$  and  $w_f$  is the weight assigned to feature  $f$ . In our experiments, we spanned multiple combinations of feature weights, confirming that the best results can be obtained when the channel feature is weighted

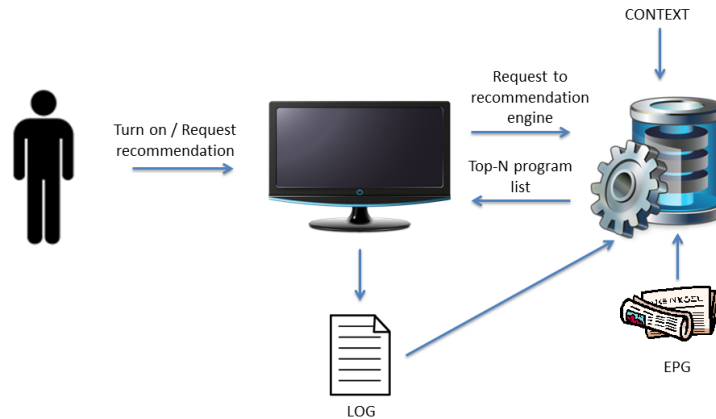


Figure 5.3: System architecture

higher than the other ones (e.g., genre). We have set  $w_f = 1$  for the features describing TV channel (i.e.,  $f \in \mathcal{C}$ ).

## 5.4 Experiment Setup

The architecture we propose for our recommender system is shown in Figure 5.3. The user interacts with a smart TV, and is allowed to request the generation of recommendations; recommendations can be generated also when the TV is turned on. The request is forwarded to the recommendation engine, that exploits the log of the past user’s synthonizations, along with the context of the user and the EPG, in order to determine the list of the top-N programs to be recommended among those currently on air. Note that some kinds of contextual information may be automatically determined by the system, like the time, but others might need to be manually declared by the user, like the people with whom she is watching TV or her current mood.

### 5.4.1 Dataset

The TV dataset used in this chapter has been collected by an independent media agency during a period of 6 months in 2013, for a total of 21 million viewing events and 56,101 EPG entries either over-the-air (digital terrestrial broadcasting) or satellite, free or pay-TV. We define a viewing event as the event of synthonization of a user to some channel, with its start and end time. The data is related to *all* the italian TV programs, spanning 24 hours per day. Our dataset has complete information on 217 channels, including the ones that have a lag version, e.g. Cartoon Network and Cartoon Network +1. The volume of data and its completeness make the challenge more difficult in comparison to other linear TV works, which usually consider only the prime time slot and the top channels.

The EPG entries describe 21,194 distinct programs (1,754 of which are TV series), classified into 7 genre categories and 117 sub-genre categories. Repeated shows as TV series or daily news are counted only once. The catalog of programs is very dynamic: on average there are 82 new programs each day (about 0.3% of the total number of

## Chapter 5. Linear TV Program Recommendation

---

**Table 5.1:** *Characteristics of TV dataset*

users	13,664
households	5,666
programs	21,194
channels	217
EPG entries	56,101
Time span	6 months
genres	7
subgenres	114
view events	21 millions

**Table 5.2:** *Demographic distributions of the users*

males	41 %
females	47 %
children	12 %
0-19	17 %
20-34	15 %
35-64	45 %
65+	23 %
mono component families	9 %
multi component families	91 %
users with children	33 %
users without children	67 %

programs). The available metadata for the programs are title, genre, and sub-genre, while for the tuning events are channel, start time, end time. The available data for the viewing events are channel, start time, and end time. All entries in the dataset (viewing events and EPG) have a one-minute resolution. A complete overview of the dataset is presented in Table 5.1 The dataset is available for download<sup>1</sup>. Demographic distribution of the users is reported in Table 5.2.

Figure 5.4 shows the distribution of audience in the top-8 Italian TV channels, on the y-axis, during the hours of the day, on the x-axis. Values report the ratio of total watching minutes in the dataset for a specific hour of day and a specific channel. The darker the color, the higher is the number of viewers. As we can see from the plot, the distribution is heavily skewed towards a limited number of channels and prime time slots.

Figure 5.5 shows the distribution of the total audience during the hours of the day. The TV viewers peak in the prime time slot. Another smaller peak is around lunch time, from 12 to 14.

Figure 5.6 reports the maximum audience share value reached by each program, reported on the x-axis. Programs are sorted by decreasing maximum share value. As we can see, only less than 1% programs in the dataset reached a share value of 20%, but many programs have a low maximum share value.

---

<sup>1</sup><http://recsys.deib.polimi.it/tv-audience-dataset>

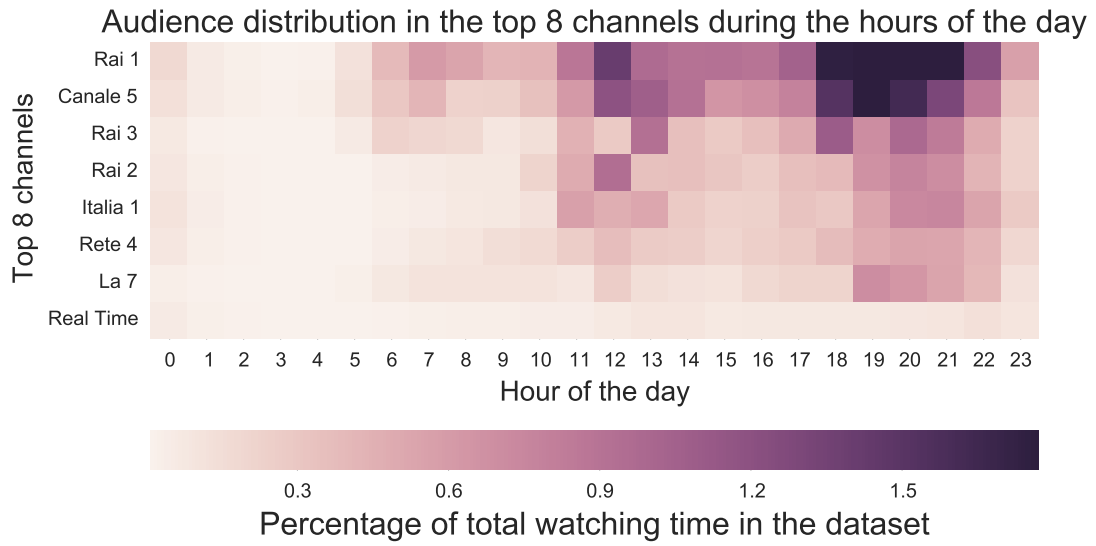


Figure 5.4: Heatmap showing the distribution of audience in the main italian TV channels

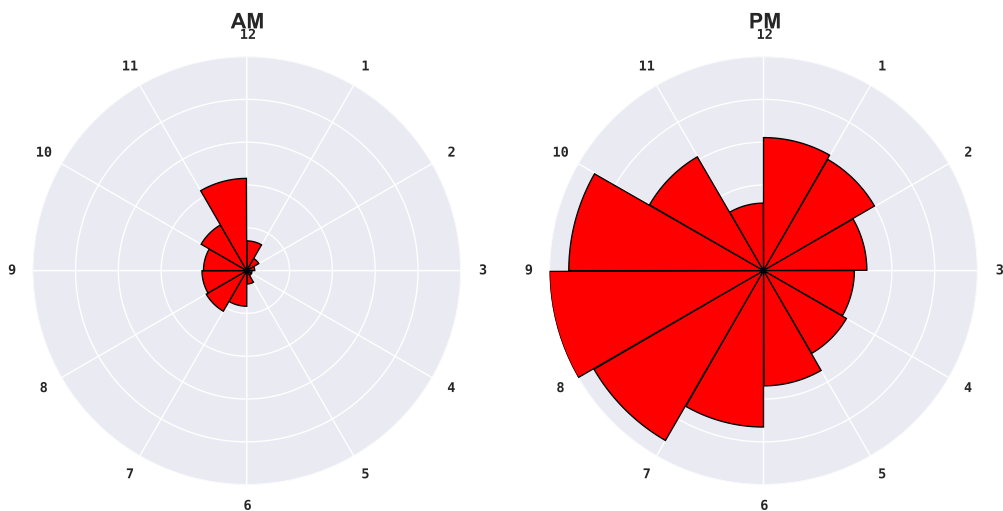


Figure 5.5: Distribution of viewers during the hours of the day

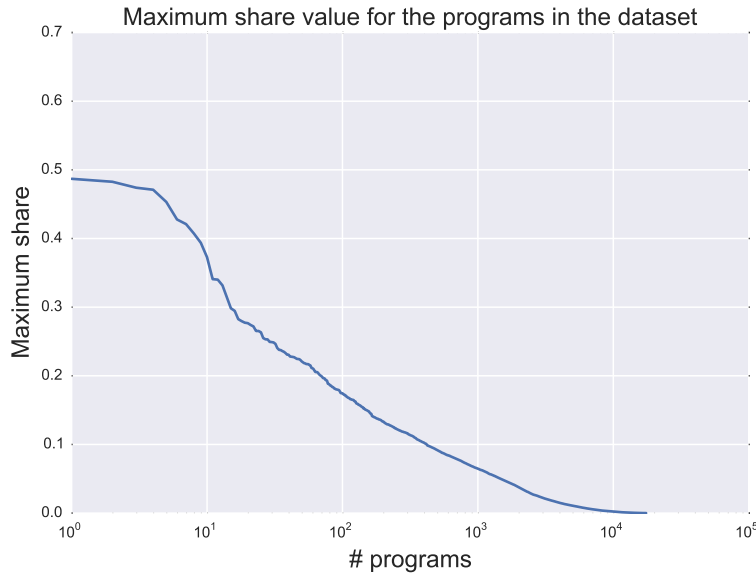


Figure 5.6: Program popularity of the dataset

#### 5.4.2 Experiment 1: Discrete Contextual Recommender

We used a subset of the dataset described in Section 5.4.1 containing 7,921 users and 119 channels. The log of program views spans from December 3rd, 2013 to March 1st, 2014, and contains 10,313,499 entries. We deemed the syntonizations shorter than three minutes as not relevant, retaining 6,525,541 log entries. Each log row specifies the identifier of the user and that of the program she watched, along with the start time, the end time and the people with whom the user watched the program. The latter three pieces of information were used to determine the values of the four context dimensions that we chose: channel, day of the week, time slot and *familiar context*, where with familiar context we mean just the people with whom the user was watching TV. More precisely, start and end time were employed to derive the day of the week and the time slot, where the available values for the time slot are shown in Table 5.3. We identified five possible relevant values for the familiar context, summarized in Table 5.4, depending on the age of the people; people older than 15 years were considered adults. The log was split in a training set, including the syntonizations between March 3rd, 2013 and February 15th, 2014 (5,438,977 entries), and a test set, containing the remaining ones (1,086,564 entries). The former was used to build the model, while the latter to assess the quality of the recommendations.

We executed our algorithm, from now on dubbed *CxtOrd*, using different combinations of context dimensions and program attributes, with the aim of evaluating their usefulness in the generation of the recommendations. In particular, we tested the non-contextual alternatives which build the user preference tensor  $R$  exploiting the sole channel and the sole genre. Then, we tried to enrich the tensor with the various context dimensions.

Our algorithm was compared also with a naive non-contextual and non-personalized methodology, dubbed *TopPop*, recommending to each user, in each context, the list of programs broadcast on the  $N$  channels that were globally the most seen.

**Table 5.3:** *Time slots*

Start time	End time	Description
02:00:00	07:00:00	Graveyard slot
07:00:00	09:00:00	Early morning
09:00:00	12:00:00	Morning
12:00:00	15:00:00	Daytime
15:00:00	18:00:00	Early fringe
18:00:00	20:30:00	Prime access
20:30:00	22:30:00	Prime time
22:30:00	02:00:00	Late fringe

**Table 5.4:** *Familiar contexts*

Familiar Context
Mixed: adults + children
Group of adults
Group of children
Adult alone
Child alone

Finally, we considered another less trivial, non-contextual and non-personalized competitor, named *ShortestTimeSinceStart*, always compiling the recommendation list with the programs started since the shortest time.

We had performed some trials also using traditional collaborative filtering. However, as explained in the introduction of the chapter, the dynamism of the item catalog makes such techniques ill-suited for TV program recommendation, and indeed the obtained results were extremely poor. Therefore, we do not show collaborative filtering results in the following sections.

All the experiments were repeated three times, considering three different compositions of the test set:

- The whole test set (7,921 users, 1,086,564 program views).
- Subset obtained excluding the users who have shown to be not very active, having watched only 7 channels or less (5,824 users, 959,141 program views).
- Subset obtained including only the very active users, having watched 28 channels or more (201 users, 51,525 program views).

The performance of our recommendation algorithm was evaluated using *Recall@N*, describing the number of test items which have been included in a recommendation list of length  $N$  computed in the instant in which the viewing of the test items started and in the context in which they have been watched.

More formally, let  $v$  be a program view in the test set,  $v_t$  the start time of the view,  $v_u$  the user that watched the program,  $v_i$  the program watched and  $v_c$  the context in which the view took place.  $TopN(u, c, t)$  is the set of top- $N$  items for the user  $u$  in context  $c$  among those on air at time instant  $t$ , determined with the recommendation methodology to be evaluated. *Recall@N* is computed as follows:

$$Recall@N = \frac{|v \in \text{Test Set} : v_i \in TopN(v_u, v_c, v_t)|}{|v \in \text{Test Set}|} \quad (5.6)$$

We executed experiments for  $N=1$ ,  $N=3$  and  $N=5$ .

### 5.4.3 Experiment 2: Smoothed Contextual Recommender

Two main factors affect the preferences of a TV user: channel and time. Indeed, the user preferences strongly depends on the temporal context (i.e., day of week and time of day). For instance, during dinner the user might be accustomed to watch newscasts, while after dinner she wants to relax and will likely opt for a movie or a reality show. It is worth noting that each user has his own habits, so, as an example, a user might eat dinner at 7PM and another one at 9PM. Furthermore, TV users are strongly affiliated with channels. The typical user switches the TV on and surfs over a very limited number of channels to select the program to watch. In many cases, the choice is restricted to the single channel the user is used to watch. Sometimes, the TV is simply switched on a default channel routinely, regardless the program currently live. As a consequence, we assume that only a minor role is played by the characteristics (e.g., genre and sub-genre) of the broadcast TV program. This assumption motivates the following three baseline algorithms, which are based solely on contextual parameters (channel and time slot) and correspond to the Discrete Contextual Recommender (CxtOrd) discussed in Section 5.3.1.

**Top channel** This baseline algorithm recommends the TV programs that are scheduled during slot  $s$  on the most popular channels. Recommendations are the same for each user. The popularity of a channel is defined in terms of the total number of watching minutes accumulated by that channel. For each channel feature  $c \in \mathcal{C}$  we compute the popularity  $r_c$  of channel  $c$  as

$$\hat{r}_c = \sum_{s,u,t} r_{ucs}(t) \quad (5.7)$$

The algorithm recommends first all the TV program scheduled during slot  $s$  on the most popular channel. If more recommendations are needed, the algorithm recommends all the TV programs scheduled on the second most popular channel during the same slot  $s$ , and so on. Partial ordering of TV programs within a channel is based on their schedule.

**Top channel per user** This baseline algorithm is a refined version of the previous one. The algorithm recommends the TV programs that are scheduled during slot  $s$  on the most popular channels, where channel popularity is computed on a per-user basis.

The popularity of a channel is defined in terms of the total number of watching minutes accumulated by user  $u$  on channel  $c$ . For each user  $u$  and for each channel feature  $c \in \mathcal{C}$  we compute the popularity  $r_{uc}$  as

$$\hat{r}_{uc} = \sum_{s,t} r_{ucs}(t) \quad (5.8)$$

**Top channel per user and slot** This baseline algorithm is a further refinement of the previous ones. The algorithm recommends the TV programs that are scheduled during slot  $s$  on the most popular channels, where channel popularity is computed on a per-user and per-slot basis.



The popularity of a channel is defined in terms of the total number of watching minutes accumulated by user  $u$  on channel  $c$  during time slot  $s$ . For each user  $u$ , time slot  $s$  and channel feature  $c \in \mathcal{C}$  we compute the popularity  $r_{ucs}$  as

$$\hat{r}_{ucs} = \sum_t r_{ucs}(t) \quad (5.9)$$

We used the full dataset described in Section 5.4.1. The dataset is partitioned into two subsets along the time dimension: training and test. The *training set* spans the first five months of the dataset. The *test set* spans the last month.

The test set is partitioned into disjoint intervals with one-hour duration. For each user  $u$  and for each interval in which the user has watched any TV program, we extract the list of TV programs scheduled in that interval. For each program  $i$  in the list, we estimate the relevance  $\hat{r}_{ui}$  according to the recommender algorithm. The list is sorted on the basis of  $r_{ui}$  and the top-N most relevant elements are presented to the user. Finally, we verify if the TV programs watched by the user in the time frame are in the list of recommended programs and we compute average rank and recall.

Both recall and average rank have been computed by weighting the TV programs on the basis of the number of minutes the user watched the program in the time interval.

The results compare three versions of our algorithm (denoted as CxtBlend) and three versions of the baseline approach (denoted as topCh).

- **topCh**: baseline algorithm (5.7)
- **topCh\_u**: baseline algorithm (5.8)
- **topCh\_us**: baseline algorithm, (5.9)
- **CxtBlend**: algorithm proposed (5.4) where genre and sub-genre weights are set to 0.
- **CxtBlend\_g**: the same algorithm as CxtBlend, where genre weights are set to 0.1 and sub-genre weights are set to 0.
- **CxtBlend\_gg**: the same algorithm as CxtBlend, where both genre and sub-genre weights are set to 0.1.

## 5.5 Results

---

### 5.5.1 Experiment 1: Discrete Contextual Recommender

In this subsection we present in a tabular form the results obtained with the experimented methodologies. Tables 5.5, 5.6 and 5.7 show, respectively, *Recall@1*, *Recall@3* and *Recall@5*. The tables are divided in two parts: the upper one shows the baselines while the lower one shows our proposed algorithm variants.

In the following the results reported in the tables are analyzed in detail, starting with the whole test set and then considering the subsets.

## Chapter 5. Linear TV Program Recommendation

**Table 5.5: Recall@1**

Algorithm	All	$\geq 8$ chan.	$\geq 28$ chan.
TopPop	19.26%	17.65%	10.28%
ShortestTimeSinceStart	3.46%	15.89%	7.34%
CxtOrd – Channel	33.04%	22.93%	10.92%
CxtOrd – Genre	13.91%	12.87%	5.85%
CxtOrd – Channel, Genre	31.57%	13.96%	6.56%
CxtOrd – Fam. Cont., Channel	33.95%	31.92%	19.33%
CxtOrd – Fam. Cont., Genre	8.56%	7.94%	3.22%
CxtOrd – Fam. Cont., Channel, Genre	33.20%	22.94%	13.96%
CxtOrd – Day, Time, Channel	<b>39.23%</b>	<b>37.02%</b>	<b>22.36%</b>
CxtOrd – Day, Time, Genre	18.11%	16.75%	1.44%
CxtOrd – Day, Time, Channel, Genre	32.83%	30.84%	19.16%
CxtOrd – Fam. Cont., Day, Time, Channel	39.11%	36.34%	21.77%
CxtOrd – Fam. Cont., Day, Time, Genre	17.38%	15.96%	7.58%
CxtOrd – Fam. Cont., Day, Time, Channel, Genre	33.29%	31.31%	18.88%

**Table 5.6: Recall@3**

Algorithm	All	$\geq 8$ chan.	$\geq 28$ chan.
TopPop	45.19%	43.17%	24.98%
ShortestTimeSinceStart	11.33%	33.03%	17.77%
CxtOrd – Channel	63.12%	41.22%	23.38%
CxtOrd – Genre	30.83%	28.78%	13.93%
CxtOrd – Channel, Genre	60.29%	35.51%	19.06%
CxtOrd – Fam. Cont., Channel	63.94%	61.38%	38.93%
CxtOrd – Fam. Cont., Genre	20.72%	19.41%	8.53%
CxtOrd – Fam. Cont., Channel, Genre	49.47%	46.44%	29.86%
CxtOrd – Day, Time, Channel	<b>67.05%</b>	<b>64.59%</b>	<b>42.11%</b>
CxtOrd – Day, Time, Genre	37.94%	35.72%	7.33%
CxtOrd – Day, Time, Channel, Genre	60.98%	58.20%	37.88%
CxtOrd – Fam. Cont., Day, Time, Channel	66.94%	62.12%	40.70%
CxtOrd – Fam. Cont., Day, Time, Genre	36.66%	34.42%	17.47%
CxtOrd – Fam. Cont., Day, Time, Channel, Genre	62.00%	59.26%	38.14%

**Table 5.7: Recall@5**

Algorithm	All	$\geq 8$ chan.	$\geq 28$ chan.
TopPop	59.93%	57.81%	31.14%
ShortestTimeSinceStart	18.07%	45.27%	25.51%
CxtOrd – Channel	77.38%	51.19%	30.64%
CxtOrd – Genre	42.07%	39.17%	19.83%
CxtOrd – Channel, Genre	65.18%	48.53%	27.72%
CxtOrd – Fam. Cont., Channel	77.93%	74.09%	51.23%
CxtOrd – Fam. Cont., Genre	73.89%	30.14%	14.09%
CxtOrd – Fam. Cont., Channel, Genre	63.17%	60.26%	39.81%
CxtOrd – Day, Time, Channel	78.58%	<b>76.56%</b>	<b>52.65%</b>
CxtOrd – Day, Time, Genre	49.82%	47.14%	15.45%
CxtOrd – Day, Time, Channel, Genre	75.31%	72.96%	48.90%
CxtOrd – Fam. Cont., Day, Time, Channel	<b>78.67%</b>	72.30%	51.00%
CxtOrd – Fam. Cont., Day, Time, Genre	48.26%	45.59%	24.32%
CxtOrd – Fam. Cont., Day, Time, Channel, Genre	76.29%	74.03%	49.67%

**Full Test Set** A first aspect which can be noticed from the results is that the differences between context-driven and baseline methodologies are larger when recommending few items. This happens because many users watch just a limited number of channels, and therefore even simple strategies are able to identify the proper program in lists containing several items.

We immediately note that the non-personalized methods TopPop and ShortestTimeSinceStart show very poor performance, while the personalized model based on the channel obtains very high recall. This suggests that the users' preferences are more important than the time elapsed since the program started to determine the right suggestion. Note also that the results for the personalized model relying on the genre are not good. The usage of the genre seems to confuse the system instead of helping; in fact, adding the genre to the model based on the channel adds noise instead of improvement.

We observe that the models with the channel behave better than those envisaging the genre, that again seems to confuse the system. The addition of the familiar context brings some improvements, but these are really small: the recall increase is less than 1% with respect to the model based only on the channel. A significant gain, on the other hand, is provided by the usage of date and time. The best-performing model – the one including day, time and channel – improves the non-contextual alternative based on the channel of 6.19% for *Recall@1*, 3.93% for *Recall@3* and 1.20% for *Recall@5*; as explained above, the shorter the recommendation list, the larger the recall increment. The addition of the familiar context to the model envisaging day, time and channel does not provide significant improvements, with the exception of *Recall@5*.

The fact that the best model is the one envisaging day, time and channel, together with the good performance shown by the non-contextual model with only the channel, suggests that the habit factor is very important in the choices of TV viewers: many users watch very often the same channels in the same time slots.

**Test Sets Obtained Excluding the Less Active Users** In this case we note that the differences between the methodologies are wide also for *Recall@3* and *Recall@5*: this happens because the users in these test sets are used to watch many channels, and so it may be difficult to capture their more dynamic behaviour. Moreover, the negative results of the models including the genre of the program are confirmed also in this case.

In general, for each model tested in the experiments, the recall value decreases with respect to that measured with the same models on the whole test set, again because these users have seen several channels and so the recommendation is more difficult. An exception is represented by the non-personalized methodology ShortestTimeSinceStart, for which the recall obtained for the active users is greater than that achieved on the full test set. This is an interesting result, and seems to suggest that the active users are more resolved in the choice of TV programs: they know what they want to watch and change the channel when they know it is starting. The other users, on the contrary, seem to proceed in a more random way among the few channels they are used to taking into account.

The two subsets of active users confirm that the context-driven strategies show better performance than the baselines. The best model is again that envisaging day, time and channel, and the increment with respect to the recommendations generated considering only the channel is even larger than that registered with the full test set. For instance,

**Table 5.8:** Comparison between algorithms. Bolded baselines (*topChxx*) are the best among baselines for the considered metric. Bolded non-baselines (*CxtBlendxx*) are statistically better than the best baseline for that metric.

Algorithm	recall			avgRank
	@1	@5	@10	@50
<b>CxtBlend</b>	0.283	<b>0.734</b>	<b>0.852</b>	<b>5.819</b>
<b>CxtBlend_g</b>	0.290	<b>0.734</b>	<b>0.859</b>	6.518
<b>CxtBlend_gg</b>	0.290	<b>0.733</b>	<b>0.858</b>	6.607
<b>topCh</b>	0.125	0.423	0.639	11.942
<b>topCh_u</b>	0.232	0.612	<b>0.797</b>	<b>7.512</b>
<b>topCh_us</b>	<b>0.282</b>	<b>0.658</b>	0.759	12.587

in the test set containing only the users having seen at least 28 channels, the increments are 11.44% for *Recall@1*, 18.73% for *Recall@3* and 22.01% for *Recall@5*.

Differently from what we observed in the experiments with the full test set, in this case the familiar context introduces a significant increment in the quality of recommendations. For instance, when the test set containing only the users having seen at least 28 channels is taken into account, the increments with respect to the model envisaging only the channel are 8.41% for *Recall@1*, 15.55% for *Recall@3* and 20.54% for *Recall@5*. However, the contribution of the familiar context is canceled when the familiar context is considered in addition to day and time. This means that the effect of the users’ habits remains stronger than the impact of the familiar context, also for the active users.

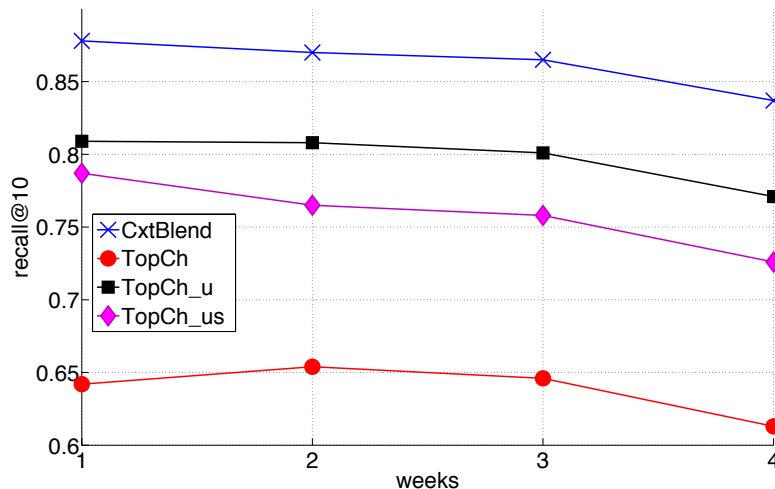
**Summary of the Evaluation**

The described experiments showed that our methodology can provide accurate recommendations to TV users relying exclusively on implicit feedback. In addition, the experiments proved that in the considered scenario the context is decisive in the recommendation process. In more detail, the day and time context dimensions showed to be relevant for all the users, while the familiar context proved not be very significant.

**5.5.2 Experiment 2: Smoothed Contextual Recommender**

Table 5.8 presents an overview of the experimental results. We omitted the results of the score aggregation algorithm (described by Equation 5.3) because they are much worse than all the others (both ContextBlend and TopCh). The best baselines are those recommending the most popular channels on a per user-basis. Including the time slot in the baseline slightly increases the accuracy of recommendations. All three versions of our algorithm outperform the baselines for all of the metrics, with the exception of *Recall@1*. We also observe that the inclusion of genre and sub-genre does not increase the accuracy of recommendations, as noted in the previous subsection.

Figure 5.7 presents *Recall@10* for the same algorithms. The recall is plotted as a function of the test week (there are four weeks in the test set). The accuracy of all the algorithms decreases as the week is more distant from the end of the training period (week 1 is the week immediately following the training period, week 4 is the most distant in time from the training period). This can be explained by a slight preference drift of users which make the prediction harder.



**Figure 5.7:** Performance decrement in terms of Recall@10 as the weeks are more distant from the training period.

**Table 5.9:** Recall@10 for ContextBlend and TopCh (baseline) for different demographic categories of users.

	CxtBlend	CxtBlend_g	CxtBlend_gg	TopCh	TopCh_u	TopCh_us
Male	0.853	0.850	0.849	0.628	0.787	0.751
Female	0.873	0.870	0.870	0.662	0.809	0.778
Children (no gender)	0.836	0.826	0.824	0.517	0.767	0.652
0-19 years old	0.825	0.816	0.814	0.515	0.758	0.640
20-34 years old	0.816	0.808	0.805	0.571	0.755	0.656
35-64 years old	0.833	0.829	0.829	0.621	0.765	0.728
65+ years old	0.917	0.916	0.916	0.703	0.856	0.844
Mono-component family	0.916	0.914	0.913	0.649	0.855	0.837
Multi-component family	0.864	0.862	0.861	0.651	0.800	0.770
User with children	0.801	0.795	0.794	0.577	0.741	0.657
User without children	0.880	0.877	0.877	0.656	0.813	0.788

Table 5.9 presents a drill-down of the *Recall@10* for different demographic categories. With respect to age, older users have a more regular viewing pattern and obtain the best recall. Families with several components are more difficult to predict. Surprisingly, female watch volume is slightly easier to predict than male one.

The lower absolute recall values with respect to the results in Section 5.3.1 are due to the different time slot definition: the coarser is the slot, the higher is the recall.

## 5.6 Discussion

We presented a new family of algorithms for the context-driven recommendation of TV programs in linear TV services. Users are modeled by taking into account the specific consumption patterns typical in the settings of TV, strongly influenced by time context and channel preferences. The drawbacks of representing time context as a discrete variable have been mitigated by introducing the notion of context distance; thus, the user model is built by aggregating nearby contextual profiles. Empirical experiments

over a large-scale linear TV dataset compare the proposed family of algorithms with a set of baseline approaches (based on channel preferences) and demonstrate a significant improvement in recommendation quality when context is considered.

This research shows that even a simple context-driven algorithm, i.e. the algorithm that recommends the aired program of past weeks on the most popular channels of the user is much more effective than traditional approaches. This is because this simple baseline takes into account seasonality effects and is not affected by the ephemeral catalog issue.

This chapter highlights that habits drive the Linear TV consumption. Intuitively, personalization leverages the user's habits and should produce high quality recommendations. However in the Linear TV domain the habits are not involving items but contexts: users are affiliated to channels and time-slots, not to programs themselves. The proposed method is, however, not a pure context-driven algorithm because it exploits past user preferences. The novelty of our approach lies in the paradigm shift that replaces items (i.e. programs) with contexts (i.e. channel and time of day). Only in the recommendation phase the contexts are actually converted to items. We hope that these results will foster research in this branch of recommendation algorithms.

---

# CHAPTER 6

---

## Linear TV Audience Prediction

---

This chapter tackles the TV audience prediction problem, describing the work presented in [98]. The challenges and peculiarities of this domain have been already introduced in Chapter 5. Predicting accurate information about TV audiences is important for broadcasters, advertisers and advertising agencies. Television networks sell time to advertisers at a price that depends on the projected television audience [134]. As an example, the average cost for a 30 seconds TV commercial in the U.S. ranges between \$100 thousand and \$2.4 million, depending on the expected audience of the TV program [42]. According to recent data presented in [66, 92], worldwide television advertising expenditure was \$178 billion in 2014 (\$78 billion in the U.S. alone) and is expected to reach \$236 billion in 2020.

The size of television audience is measured either in terms of *television ratings* (percentage of households tuned to a specific TV station) or *rating points* (one ratings point represents 1% of viewers). Rating points are the currency for the TV commercials. As an example, in the U.S. the average value of one rating point is \$780 million per year and can be as high as \$70 thousand per minute [94].

Television advertising time is purchased some time in advance, with advertisers planning to achieve a target number of ratings points during their commercials [68, 73].

Hence, accurate ratings forecasts are vital to both advertisers and TV networks, with a difference of just one rating point resulting in a substantial gain or loss for either a broadcaster or an advertiser [88, 51].

Context in this setting is the driving force for the prediction: TV ratings for each channels, which are by themselves contextual variables, have a strong seasonal component that traditional collaborative approaches are not able to model correctly.

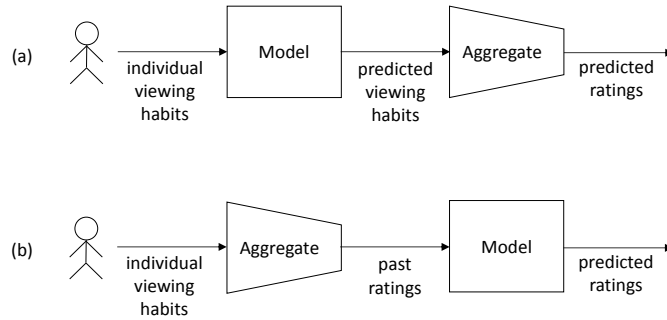


Figure 6.1: Data aggregation and alternative models

## 6.1 Problem Definition

---

There are two possible approaches to the problem of predicting television ratings, differing on the granularity of the data used by the models [88].

- Algorithms in the first family build a model to estimate future viewing habits from measurements, which are later aggregated into television ratings (Figure 6.1.a). Examples are collaborative-filtering algorithms for the watch–next problem [61].
- Algorithms in the second family aggregate measured viewing habits into historical television ratings which are later used to build a model (Figure 6.1.b). Recent works show that data aggregation simplifies non linearity and improves accuracy of predictions [88].

The size of television audience is described in terms of television ratings. Television ratings can be measured either with *rating points* (percentage of households tuned to a specific TV station) or with *viewing time* (average number of minutes a household is tuned to a specific TV station). It is always possible to convert rating points in viewing time, and viceversa, by knowing the number of households in the sample.

Throughout the rest of the chapter we will use the term **ratings** with the meaning of **television ratings**, measured in terms of average **viewing time** per user and channel (the viewing time is an implicit measurement of the user interest in a specific program).

Rating forecasts are based on historical measurements of TV viewing habits. Measurements rely on TV meters installed in a representative sample of households by independent media companies (e.g., Nielsen, Auditel, AGF, BARB) As an example, in 2014 Nielsen was using a sample of more than 5,000 households, containing over 13,000 people, to be representative of 116.3 million TV homes in the U.S. [94]. Recently, some works suggested the usage of social networks as an alternative way to measure television ratings [37][129].

In this Chapter we address the problem of predicting TV ratings based on users’ viewing habits and on some characteristic of the TV programs. The raw input data, described in Section 5.4.1, to the system are:



1. the EPG (Electronic Programming Guide) containing program-id, genre, sub-genre, start-time, end-time, and TV channel for all the scheduled programs;
2. the set of viewing events containing house-id, start-time, end-time and TV channel for all the viewing events (e.g., whenever a user watched a TV channel).

In order to have a data structure easier to manage, we pre-process and simplify the raw data. We divide the time span of a day into time slots of equal duration  $L$ .

The main goal is, given user  $u$  and time slot  $s$ , to estimate the number of minutes  $\hat{r}_{us}$  (e.g. the rating of the user) for all the programs in time slot  $s$ . We use the same notation introduced in 5.1.1.

## 6.2 Background

---

A number of papers address the watch–next problem, which aims to provide recommendations for the next program to watch following the currently watched program. They are covered in Subsection 5.2. However, there is a lack of recent research works on the problem of predicting television ratings, mainly because most of the datasets are proprietary. The few papers in the literature describe algorithms that use static regression models, with covariates based on attributes of programs and time of viewing events [88, 87]. These algorithms give static predictions as the forecasts do not depend on the prediction horizons chosen, but only on covariates [43, 42]. What is missing from previous work is the development of models reflecting the *lead-in* effect, i.e., the fact that television programs inherit viewers from the immediately preceding programs scheduled on the same TV channel [124].

Moreover, the results are usually validated only on 5 or fewer TV channels and, in most cases, only on prime-time programs (e.g., between 20:30 and 22:30).

However, modern TV environment scenarios have hundreds of TV channels. As an example, a recent study by Nielsen shows that the typical household in the U.S. has almost 200 channels to choose from [94].

Meyer and Hyndman [88] investigate the effect of aggregation in relation to the accuracy of television rating forecasts. Models are fitted using neural networks, decision trees and regression. They show that data aggregation helps to simplify non linearities in the TV watching behaviour by averaging its fluctuations.

Danaher et al.[43] describe a two-steps logistic model: the first step models the decision to switch the TV on (viewer availability), and the second step models channel selection (program choice). The model is validated against a Bayesian averaging model, focusing on the 5 most popular TV channels during prime time (8:30PM–10:30PM).

Nikolopoulos et al. [95] show that multiple regression models outperform the item-based nearest neighbors approach when forecasting TV ratings for 12 sport events (test set) based on the ratings of 34 similar events (training set).

Kelton et al. [69] and Reddy et al. [107] design models for the optimal scheduling of TV programs in order to increase future TV ratings.

Patelis et al. [87] describe a first attempt to use dynamic models to predict TV ratings: ratings are predicted finding weeks that have the same type of competitiveness with the one to forecast, while a time series approach is used to model total viewership.

The focus of this study is on the development of models able to handle thousands of TV programs and hundreds of TV channels, and reflecting the fact that observations

close together in time are more closely related than observations further apart (lead-in).

We describe and compare different *time-series* dynamic models for the prediction of television ratings. The predictions are based on historical viewing habits, past and future schedules of TV programs, attributes of programs, and time-related contextual information, such as hour and day. We use exogenous variables to capture the intrinsic appeal of new TV programs from observed characteristics of past TV programs with similar attributes.

Our dynamic models are compared with three state-of-the art approaches, (i) context-aware collaborative-filtering with fuzzy seasonal context described in [61]; (ii) static regression models with program-based covariates (e.g., genre, sub-genre) and time-based covariates (e.g., day of week, hour of day) described in [43]; and (iii) nested logistic model described in [42].

### 6.3 Dynamic Models

---

When dealing with time series analysis, the theory of stationary stochastic processes becomes a powerful tool to analyze and predict the main dynamics of a signal. In our case, the TV ratings that we observe can be seen as a realization of a stochastic process  $R(t)$  assumed to be sum of three components [132]

$$R(t) = Z(t) + T(t) + S(t), \quad (6.1)$$

where  $Z(t)$  is a *Stationary Stochastic Process* (SSP),  $T(t)$  denotes a trend (e.g.  $T(t) = \alpha t + \beta$  would be a linear trend) and  $S(t)$  describes a nonrandom cyclic influence like a seasonal component. The additive structure is appropriate (rather than the multiplicative one) because the seasonal variation does not vary with the level of the series [91].

The innovative feature of this representation is that  $Z(t)$  can be a function of its past values, e.g.  $Z(t-1)$ ,  $Z(t-2)$ , etc.

In order to predict  $R(t)$ , one first needs to provide the mathematical description of the three components. In particular, the identification process is divided into two steps: the identification and removal of  $T(t)$  and  $S(t)$ , representing the deterministic parts of the process, and the identification of the model structure of the stochastic part of the process, namely  $Z(t)$ .

#### Trend and seasonality removal

The trend identification  $\hat{T}(t)$ , by assuming a linear trend, can be performed by minimizing via least squares the sum of residuals

$$\min_{\alpha, \beta} \sum_{i=1}^N [r(t) - (\alpha t + \beta)]^2. \quad (6.2)$$

where  $r(t)$  are the observed ratings in  $R(t)$  and  $N$  is the total number of observations. In the de-trended series  $R_{ZS}(t) = R(t) - \hat{T}(t)$ , the seasonal component  $S(t)$  can be estimated as

$$\hat{S}(t) = \frac{1}{(N/\tau) - 1} \sum_{j=1}^{(N/\tau)-1} r(t + j\tau) \quad (6.3)$$

where  $\tau$  is the period of the seasonality, and  $(N/\tau) - 1$  is the total number of observations in a single period.

The period  $\tau$  can be easily detected by analyzing the Fourier transform of the signal (in our case, a peak at a frequency corresponding to 7 days would be a weekly seasonality).

Once the seasonal part is removed with

$$R_Z(t) = R_{ZS}(t) - \hat{S}(t) = R(t) - \hat{T}(t) - \hat{S}(t) \quad (6.4)$$

an estimation of  $Z(t)$  can be obtained from the data  $R_Z(t)$ , as explained in the next section. Elements  $r_Z(t)$  of  $R_Z(t)$  are the television ratings once trend and seasonality have been removed.

#### AR model identification

In this chapter, Auto-Regressive (AR) models are selected for the estimator of  $Z(t)$  as they are simple and linear in the parameters. In what follows, we will also show that such a model class is flexible enough to capture the main TV rating dynamics. Formally, an AR model is defined as

$$r_Z(t) = \varphi(t)' \theta + \xi(t) \quad (6.5)$$

where  $\theta = [\vartheta_1 \ \vartheta_2 \ \dots \ \vartheta_{n_\vartheta}]'$  is a vector of unknown parameters,  $\varphi = [r_Z(t-1) \ \dots \ r_Z(t-n_\vartheta)]'$  is the regressor with past data and  $n_\vartheta$  is the model order. In simple words, an AR model is a linear regression over the past samples of the time series at hand, after removing trend and seasonality.

The *Minimum Description Length (MDL)* criterion can be used to estimate the model order. It relies on the “*modeling by shortest data description*” principle [113] and gives consistent estimates of the order of AR models, assuming that the number of samples is much larger than the number of estimated model parameters (which is always the case in this application). The MDL formula reads as follows:

$$MDL(n_\vartheta) = (\log n_\vartheta) \frac{n_\vartheta}{N} + \log \left( \frac{1}{N} \sum_t \epsilon(t)^2 \right) \quad (6.6)$$

where  $\epsilon(t) = r_Z(t) - \varphi(t)'\hat{\theta}$  is the error between the measured value and the model output. An alternative method is cross-validation. For this, we need to divide the time series in two parts, and use only the first one to identify the parameters while the second one assesses the model quality to establish the correct model order.

Once the model order is determined, the model parameters can be estimated according to the prediction error method [79]. Following such an approach, the best model using  $N$  data (parameterized by  $\hat{\theta}_N$ ) is the one minimizing the output prediction error, that is

$$\hat{\theta}_N = \arg \min_{\theta} \frac{1}{N} \sum_{t=1}^N [r_Z(t) - \varphi(t)'\theta]^2. \quad (6.7)$$

The resulting  $n_\vartheta$ th order AR model can be rewritten as a first-order,  $n_\vartheta$ -dimensional vector autoregression model:

$$\begin{bmatrix} r_Z(t) \\ r_Z(t-1) \\ \dots \\ r_Z(t-n_\theta-1) \end{bmatrix} = \begin{bmatrix} \vartheta_1 & \dots & \vartheta_{n_\theta} \\ 1 & 0 & 0 \\ \dots & \dots & \dots \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_Z(t-1) \\ r_Z(t-1) \\ \dots \\ r_Z(t-n_\theta) \end{bmatrix} + \begin{bmatrix} \xi(t) \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

or, in matrix notation,

$$R_Z(t) = AR_Z(t-1) + E(t)$$

with obvious definitions for  $A$  and  $E(t)$ . Finally, the  $K$ -step predictor of the stochastic component  $Z$  is

$$\hat{R}_Z(t+K|t) = A^K R_Z(t),$$

whereas the total predicted TV rating reads

$$\hat{R}(t+K|t) = \hat{R}_Z(t+K|t) + \hat{T}(t+K) + \hat{S}(t+K). \quad (6.8)$$

where  $K$  is the number of steps in the future (in our case, days) for the prediction of TV ratings.

#### ARX model identification

When some exogenous variables act on the dynamics of the process, the analysis of the time series alone is not able to describe the underlying data generation mechanism nor to accurately forecast future values of the time series. This is the case when we want to include information about which kind of program is broadcast (genre and sub-genre) or whether the considered period is of vacation or not. **Exogenous** inputs in our auto-regressive model are equivalent to **side information** in collaborative-filtering.

Exogenous inputs can be easily embedded into AR models by considering their ARX extension (where ARX stands for Auto-Regressive models with eXogenous inputs). Formally, an ARX model is defined as

$$r_Z(t) = \varphi(t)' \theta + \xi(t) \quad (6.9)$$

where  $\theta = [\vartheta_1 \ \vartheta_2 \ \dots \ \vartheta_{n_\theta}]'$  is a vector of unknown parameters, but  $\varphi = [r_Z(t-1) \ \dots \ r_Z(t-n_\theta), u(t) \ \dots \ u(t-n_u)]'$  now contains both exogenous input  $u$  (i.e., genre, sub-genre, ...) and measured ratings  $r_Z$ , with  $n_u$  being the degree of the exogenous part.

We fix  $n_u = 0$ , since we assume only algebraic dependency between input and output. This observation comes from the nature of the system at hand: e.g. we suppose that the program broadcast at a given time does not affect the future rating. Once the additional parameter  $n_u$  is fixed, the procedure remains the same as the one given for AR models.

The resulting  $n_\theta$ th order ARX model can be rewritten as a first-order,  $n_\theta$ -dimensional vector autoregression model:

$$R_Z(t) = AR_Z(t-1) + U(t) + E(t)$$

where  $U(t) = [u(t), 0, \dots, 0]^T$ . The corresponding  $K$ -step predictor of the stochastic component  $Z$  is

$$\hat{R}_Z(t+K|t) = A^K R_Z(t) + U(t),$$

as  $U(t)$  is assumed to be perfectly known, whereas the formula for the total predicted TV rating is again (6.8).

## 6.4 Experiment Setup

In this section we describe the compared methods and their settings.

### 6.4.1 Compared Methods

The dynamic models are compared with three static state-of-the-art approaches, one based on individual viewing habits and two on aggregated television ratings.

**CARS:** we implemented the context-aware collaborative-filtering model with fuzzy seasonal context described in [61]. The model is solved by using the implicit tensor-based factorization algorithm iTALS described in [60]. We first define the *user-context-item* tensor  $R = \{r_{ufs}\}$  where  $r_{ufs}$  is the total number of minutes that user  $u$  spent watching programs with features  $f$  during time slot  $s$

$$r_{ufs} = \sum_t r_{ufs}(t) \quad (6.10)$$

The tensor is approximated as the product of three latent factor matrices  $A \in \mathfrak{R}^{l \times u}$ ,  $B \in \mathfrak{R}^{l \times i}$  and  $X \in \mathfrak{R}^{l \times s}$  (one for each dimension,  $l$  is the number of latent features)

$$r_{ufs} \approx \sum_l \alpha_{lu} \cdot \beta_{li} \cdot \gamma_{ls} \quad (6.11)$$

where  $\alpha \in A$ ,  $\beta \in B$  and  $\gamma \in X$ . The estimated television ratings for user  $u$  on item  $i$  in time slot  $s$  is

$$\hat{r}_{usi} = \sum_l \alpha_{lu} \cdot \beta_{li} \cdot \gamma_{ls} \quad (6.12)$$

**STAT:** a static regression model considering program-based covariates (e.g., genre, sub-genre) and time-based covariates (e.g., day of week, hour of day) described in [43]. This model predicts the audience by a linear combination of the covariates in input. We created a model for each time series, thus resulting in  $24 * 217 = 5208$  regressors. This method does not consider any temporal dependency of the data, as each data point contributes independently to the coefficients estimation. This method is an aggregated method and requires aggregated data in input.

**NESTED:** the nested logistic model described in [42]. This model is nested in the sense that the prediction is computed by the product of two parts: the first part computed the probability of watching TV, while the second part computes the probability to watch some channel provided that the TV is being watched. The estimation is performed using weighted least squares. This model belongs to the static category and requires aggregated data. The set of covariates used is the same as the aforementioned STAT method. Like the previous method, this algorithm does not consider any temporal dependency of the data.

### 6.4.2 Settings

We used the full dataset described in Section 5.4.1. The six months of the dataset have been partitioned into three subsets: training set (first four months), validation set (one month) and test set (last month).

Based on the results presented in [43], we have chosen to apply the AR model (6.1) to each channel  $c$  and to each time slot  $s$  by defining

$$r(t) = r_{cs}(t) = \sum_{f \notin c, u} r_{uf_s}(t) \quad (6.13)$$

where  $r_{cs}(t)$  contains the television ratings per channel and slot. Overall we created  $217 \times 24 = 5,208$  models (one per channel and time slot). The models can be easily trained in parallel, each one on a different partition of the dataset.

The sampling time  $t$  has been set to one day. Smaller sampling times (e.g., one hour) have been avoided because of the lead-in anchoring effect: with one-hour sampling time the model overfits the ratings of the previous hour.

Ratings (6.13) have been detrended by using (6.2) with  $\alpha = 0$ . Frequency analysis detected a weekly seasonality, as expected, which has been removed by using (6.4) with  $\tau = 7$ .

For each channel and time slot we have applied the MDL formula (6.6) to identify the model order  $n_{\vartheta}$ . The MDL has been computed on the validation set, once each model has been trained on the test set. Table 6.1 shows that most of the models are first order models.

Order $n_{\vartheta}$	Percentage
1	75.3 %
2	10.3 %
3	5.4 %
4	2.8 %
5	1.1 %

**Table 6.1:** Orders of the auto-regressive models (only the most frequent)

We used 9 exogenous inputs  $u$  as side information in the ARX model (6.9): seven for *genre* (“young”, “sport”, “movies”, “life-style”, “entertainment”, “news”, “undefined”), one for *holiday*, and one for *live-event*. Each of them is a binary input: 1 if the attribute is present, 0 otherwise. The genre inputs are later normalized with respect to the number of non zero genres in the input. The holiday input is set to 1 if the day is a national vacation other than Saturday or Sunday. The live-event input is set to 1 if the program scheduled in the time slot is a live event. The root mean squared error (RMSE) has been used to measure the accuracy of the different methods. As we are measuring television ratings in terms of average number of viewing time, we normalize the error with respect to the number of users in the sample (5,666), in order to make it independent of the population size

$$e = \frac{\sqrt{\frac{\sum_{c,s,t} [r_{c,s}(t) - \hat{r}_{c,s}(t)]^2}{N_{\text{channels} \times \text{slots} \times \text{days}}}}{N_{\text{users}}}} \quad (6.14)$$

Error	AR	ARX	STAT
per household and channel [min/hour]	0,8218	0,8039	0,8670

Table 6.2: Error for predictions at 1-day distance

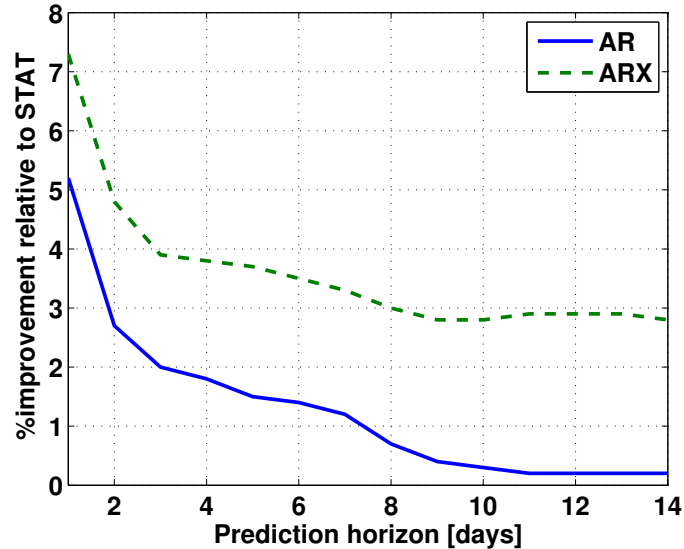


Figure 6.2: Improvement with respect to STAT method

The error  $e$  is measured in terms of

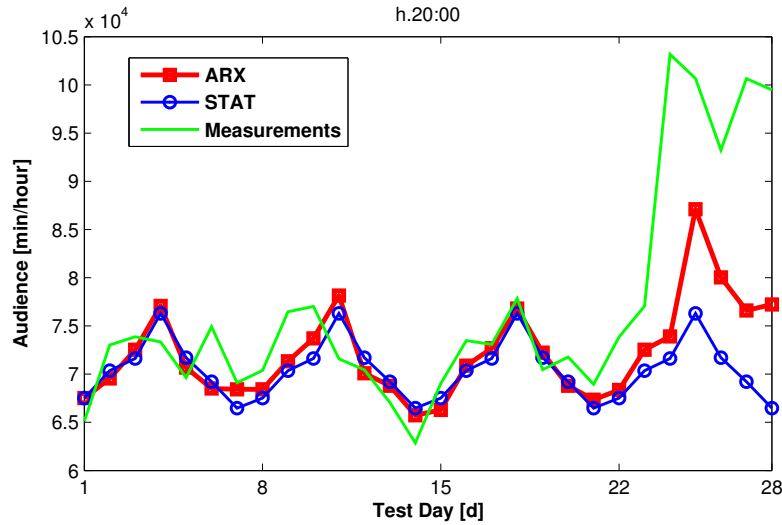
$$\frac{\text{minutes}}{\text{hour} \times \text{household} \times \text{channel}}$$

With a preliminary set of experiments we compared the accuracy of the baseline methods CARS, STAT and NESTED. The STAT method, despite its simplicity, outperformed both the CARS and NESTED methods. This can be explained by considering that, when aggregating television ratings (either measured or predicted), the seasonal effect becomes stronger and context becomes the main effect.

## 6.5 Results

Table 6.2 shows the error for the STAT, AR and ARX methods when predicting television ratings at one day of distance. The first row of the table shows the value of  $e$  for the three different models. The errors seem relatively small for all the models. But we should keep in mind that these errors have to be scaled with respect to the number of households.

Figure 6.2 shows the improvement in accuracy with respect to the static method, as a function of the prediction horizon. As expected, the dynamic models perform much better than the static one for short-term predictions (almost 8% improvement for the ARX model). When the prediction horizon increases, the accuracy of the simple AR models approaches the accuracy of the static model. However, the ARX model with exogenous inputs performs significantly better than the static model even for predictions up to 14 days in the future with an improvement of 3%.



**Figure 6.3:** Predicted vs. measured television ratings, for the most popular channel, between 20:00 and 21:00.

Figure 6.3 shows the predicted ratings for the most popular TV channel in the test month, by using both the ARX and STAT methods. Predictions are computed by using the previous three months for training, and are compared with the measured ratings. From the figure we observe that during the last week the television ratings deviated from the “normal” behavior. The static model is not able to react to this change, while the dynamic model better adapts to the new trend.

## 6.6 Discussion

The analysis of the results presented in the previous section suggest a number of interesting considerations: *(i)* models working on aggregated television ratings have a better accuracy than models working directly on viewing events, *(ii)* dynamic models have a better accuracy than static models for short term predictions, and *(iii)* dynamic models with exogenous input have better accuracy even for long-term predictions.

1. Context-aware collaborative filtering based on tensor factorization is consistently the worst performer, even with respect to less detailed and sophisticated methods. Given the fact the the algorithm is working on detailed and non-aggregated viewing data, we did not expect this result. This may have to do with the fact that aggregated viewing present strong seasonal effects that are easily described with linear models.
2. Dynamic models outperform static regression models for short term predictions. This is not surprising as dynamic model are able to quickly react to lead-in effects or to other unexpected changes in the values. With long-term predictions, auto-regressive models reduce their advantages over static models as they asymptotically became equivalent to static models.
3. Exogenous inputs are able to easily capture features of television programs and to boost the dynamism of the model even on long-term predictions, when the



correlation with past ratings is less strong.

With the billions of dollars spent annually on TV commercials, reliable audience predictions are required to evaluate the effectiveness of this investment.

In this chapter we explore a new context-driven approach for the prediction of television audience (e.g., viewing time) based on time series analysis and auto-regressive dynamic models. The inclusion of side information as exogenous variables is able to capture the intrinsic appeal of each program based on observable attributes. The dynamic models outperform collaborative-filtering and regression-based models in predicting television ratings for short and long term horizons.

We have validate our results on a large-scale dataset containing 21 million TV viewing events collected from 5,666 households over 217 channels during a period of 6 months. The dataset has been released publicly.

The superior performance of auto-regressive models with respect to context-aware collaborative filtering techniques suggests that context-driven auto-regressive models could be efficiently applied to more traditional recommender systems domains as a valid alternative – or as an integration – to collaborative-filtering, especially in the case of time-dependent recommendations.

In this domain seasonality and ephemeral catalog properties break down the ImP assumption and make traditional non-context-driven approaches to perform poorly. We showed that even simple context-driven baselines outperform complex traditional techniques and we developed context-driven algorithms that take advantage of the “context-drivenness” of the TV domain to deliver more accurate audience predictions.



---

# CHAPTER 7

---

## E-Tourism

---

The E-tourism domain exhibits characteristics that make it very different from other recommendation domains (see Figure 1.1). Item consumption is very much driven by the season: some hotels are best picks in the hot season and some of them are the best in the cold season. Weather and season can also influence the choice of restaurants as well as flight bookings. The item consumption is also driven by the intent of the user: if the user's intent is to have a very relaxing holiday enjoying the sea, this can lead to very different choices with respect to a user interested in art and museums.

The most important peculiarity in this domain is relative to the bounded capacity, e.g. the items cannot be consumed by an unlimited number of users (such as in the movie recommendation domain), but each item has a maximum capacity over which the item is not available and thus not recommendable. This property holds for seats in restaurant or in a flight as well as for rooms in a hotel. During the high season it could be difficult for a user to find a room, since most hotels are already booked. Experience tells us that hotels which are the best in the users' opinion are the first to be reserved and, consequently, the first to become unavailable in high season. We refer to this scenario as the missing short head scenario [33]. Predicting the order in which the hotels become fully booked is of great interest for the E-tourism domain. First it helps into identifying the "best" hotels and thus improve the quality of recommendation. Second, forecasting the availability of hotels can help users in planning their trips more accurately. For example a hotel booking platform may advise the user that the hotel she is seeing is going to be fully booked soon. This feature exists in real products. In this chapter we investigate the problem of ranking the hotels, from the first that becomes "unavailable" to the last available one. We show that by using contextual information we are able to outperform the baseline that uses only collaborative information.

## 7.1 Problem Definition

---

Many studies (e.g., [46, 52]) pinpoint that users' opinion is influenced by a number of factors, the predominant being others' opinion and judgments, manifested for example in on-line reviews. The issue is how to operationalize this behavior in terms of measurable factors. Intuitively, average ratings and number of reviews can both influence users' booking choices. According to the *TopAvg* method defined in [35], the hotels most likely booked by users are the ones with the largest *average rating*  $\bar{r}_i$ , defined as

$$\bar{r}_i = \frac{\sum_u r_{ui}}{n_i} \quad (7.1)$$

where  $r_{ui}$  is the rating of user  $u$  to item  $i$ , and  $n_i$  is the number of users who rated item  $i$ . However, average ratings computed over a larger support are considered more reliable by the users compared to the ones calculated over a smaller support. Therefore, *popularity* can be used as an alternative metric to define the best hotels, as with the *TopPop* method described in [35]. The two metrics (average rating and popularity) are not necessarily correlated, as low popularity may come along with a high hotel rating and vice versa.

In this chapter we aim at developing a context-driven hotel recommendation algorithm which is able to suggest the best available hotels by using average rating, popularity and other contextual signals.

## 7.2 Background

---

A first study relative to users in different availability conditions has been carried out in [34, 33]. The authors performed off-line and on-line experiments evaluating the performance of personalized and non-personalized algorithms in two availability conditions: full availability (low season) and partial availability (high season).

They proved that popularity-based methods outperform personalized methods in condition of full availability (high season). This means that the top popular hotels are the best choice for most of the users. In low availability conditions, instead, the personalized approaches are better: when the “best hotels are gone”, the taste of the user drives the decision making process. This demonstrates that context (in this case the availability of items) has a crucial impact on the recommendation performance.

In order to simulate high and low season conditions, they developed a recommender system for recommending available hotels. Modeling hotel availability allows the system to determine the “short head” of the best hotels, and so the ones that are the first to be fully booked. This recommender combines the average rating and the popularity of the hotels, following the principle that hotels that are both top rated and rated by many users are the best ones.

In this chapter we show how the performance of this recommender can be improved by taking context into account.

## 7.3 Context-Driven Available Hotel Recommender

---

Our Context-Driven Available Hotel Recommender (CDAHR) is a recommendation algorithm that takes into account average rating, popularity, stars of the hotel and con-

textual signals such as price, distance from the center and occupancy. Average rating, popularity, and the stars are, intuitively, good predictors for the hotel availability. Besides these predictors, contextual signals also play an important role. Price is a valuable information because usually good hotels are more expensive. On the other hand users tend to book hotels that less expensive. The distance from the center usually has an impact in the perceived quality of the hotel: hotels that are closer to the center tend to be booked first. The occupancy of the city is also a valuable piece of information because it could help the model in distinguishing the different training conditions.

By using these features we frame the problem as a classification problem where the dependent variable is the class available/not available. In order to do so we use large margin classifiers (Support Vector Machines) which are able to deliver accurate predictions even with a small amount of training data. In the prediction step we used the confidence in the prediction for the “available” class to rank the hotels.

## 7.4 Experiment Setup

In order to measure effectiveness of CDAHR in predicting the booking behavior of a community of users, we measure if our recommender is able to predict the classification of hotels according to their availability. We used Venere.com APIs to download the full list of hotels for a number of European cities. Moreover, for each city we identified a low-availability period with the following experiment: on a Friday we simulated last-minute reservations by querying Venere.com on available rooms for that same weekend and requiring a two night reservation for two people on all the hotels for that city. For each city we used 240 queries as training data and 20 queries as test data with different occupancy levels. For each hotel the APIs of Venere.com return the average rating, the stars of the hotel, the number of users that rated the hotel and the availability of that hotel according to the query parameters (date and rooms), as well as other contextual information such as the minimum and maximum prices, the proximity to the city center, price in high and low season conditions. For each city

- we create a ranked list of hotels according to our recommendation algorithm
- we define the occupancy  $b$  as

$$b = \frac{\# \text{ unavailable hotels}}{\# \text{ total hotels}} \quad (7.2)$$

- we define short head hotels as those ranked among the  $b$  topmost hotels in the list.

Our hypothesis is that hotels in the short head are not available, while hotels in the long tail are available.

Figure 7.1 depicts all the possible combinations of actual/predicted available/unavailable hotels. True positives (TP) are the available hotels that are in the long tail, while false negatives (FN) are the available hotels in the short head. True negatives (TN) are fully booked hotels that are in the short head, while false positives (FP) are unavailable hotels in the long tail.

In order to measure the quality of our classifier we use accuracy

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (7.3)$$

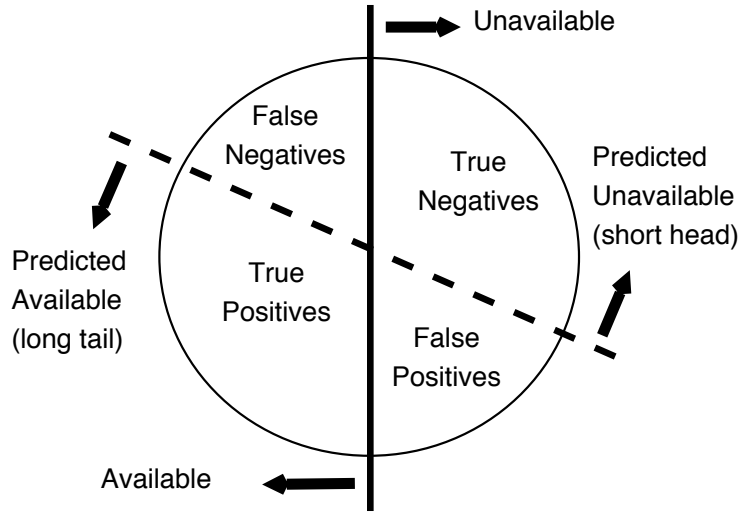


Figure 7.1: Available and not available hotels

#### 7.4.1 Compared Methods

As a baseline, we use a random classifier. Given an occupancy level of  $b$ , we can think that a random classifier performs a random permutation over the list of hotels, which is split in two parts with a proportion of  $b$ . Intuitively, the greater the proportion of hotels that are fully booked is, the more likely a random permutation will have a fully booked hotel in the first  $b * \#total$  hotels. The probability that a hotel classified as fully booked (probability  $b$ ) is actually fully booked is  $b*b$ , giving us the probability of the True Negative (TN), while the probability that is not fully booked is  $b * (1 - b)$ , representing the false negative rate. The same reasoning can be extended to the TN and FN:

$$\begin{aligned} \text{TN} &: b * b \\ \text{FN} &: b * (1 - b) \\ \text{FP} &: (1 - b) * b \\ \text{TP} &: (1 - b) * (1 - b) \end{aligned}$$

The sum of TP, TN, FP and FN is 1, either by calculating the sum of the previous definitions or by considering that together they constitute the whole space of possible events, as depicted in Figure 7.1. So the accuracy of the random classifier, given an occupancy level of  $b$  is:

$$\text{random accuracy} = b^2 + (1 - b)^2 \quad (7.4)$$

We use the *shrunk average rating* defined in [34, 33] as our baseline. This scoring function combines popularity and average rating as the following:

$$\bar{r}_i = \frac{\bar{r}_i n_i + k n_i}{n_i + k} \quad (7.5)$$

where  $k$  is a shrink constant that controls the support of the estimation. For  $k = 0$  hotels are ranked according to the traditional definition of average rating (TopAvg). For  $k \rightarrow \infty$  hotels are ranked according to their popularity (TopPop). For intermediate

values of  $k$  both popularity and average rating contribute to the definition of the short head of fully booked hotels. We used the training set to tune  $k$  for the shrank average rating for each test case.

For our CDAHR, we trained and compared an SVM classifier per each city, with different occupancy levels. We used a linear kernel and we pre-processed the input data by mean-variance normalization. We used a value of 0 as ground truth for a fully booked hotel and 1 for an available one and we used the confidence in the prediction as the score for ranking. We used a linear kernel, with  $\epsilon = 0.1$ . For each classifier we ran a 5-fold cross-validation to set the best  $C$  and  $\gamma$  values.

## 7.5 Results

Table 7.1 shows the performance of the compared methods for each city (that yields a different occupancy level). The table reports the number of hotels of each city as well as the average, minimum and maximum occupancy levels  $b$  together with the accuracy of the compared methods. First of all it is worth noting that this table validates our theoretical calculation of the random performances, because the differences from the theoretical performance are very small. Our CDAHR method outperforms the baselines in all test cases, both in low and high season scenarios (low and high levels of  $b$ ). The increment is more evident when the shrank rating yields performance that are comparable to the random baseline, e.g. London. This pinpoints that considering only rating and popularity is not sufficient for predicting the ranking among the hotels. Our context-driven method, instead, shows remarkably good performance, validating our intuition that context is the main driver for the hotel decision making process.

Table 7.2 shows the weights for the linear SVM classifier for each city. A positive sign of the weight means that that variable positively contributes to the estimation of an available hotel. A negative sign, instead, indicates that that variable makes the hotel fully booked. As a rule of thumb, a negative weight means that that variable is a characteristic of the hotel that makes it one of “the best” hotels in that particular city. We can note that the values are consistent among cities. It is important to point out that even if the occupancy is the same for each test case, it varies in the training, so it was included as feature to let the algorithm model different occupancy levels. Occupancy has a negative effect on the prediction, this means that the confidence of the algorithm in classifying a particular hotel as available decreases with the occupancy. Although the coefficient for the rating has a low absolute value, its sign is always negative: it means that the higher the average rating, the lower are the chances that the hotel is available. The same holds for proximity, the closer is the hotel, the less likely it will be available, although in this case the absolute value is lower and we can deem this variable as insignificant. Surprisingly, the popularity has a positive effect in the availability of the hotel: the more popular it is, the more likely it will be available. This can be due to the lack of correlation between popularity and average rating: a hotel with many reviews is not necessarily a good one. The stars of the hotel also have a positive effect on the availability: this can be related to the price, the higher the number of stars is, the higher is the price, and thus fewer people can afford to book that hotel. The prices in low and high season deserve a separate analysis, because they are correlated to each other. In general, a high price in high availability condition does not increase the willingness of

**Table 7.1:** Accuracy of the compared methods for each city. Number of hotels and minimum, maximum and average occupancy levels are reported.

City	Characteristics				Accuracy		
	#hotels	avg b	min b	max b	Random	Shrank	CDAHR
Moscow	129	0.302	0.279	0.426	0.583	0.695	<b>0.727</b>
Athens	246	0.295	0.276	0.313	0.603	0.681	<b>0.848</b>
Stockholm	178	0.220	0.096	0.365	0.674	0.754	<b>0.810</b>
Monaco	458	0.519	0.319	0.646	0.529	0.606	<b>0.769</b>
Amsterdam	308	0.356	0.214	0.909	0.614	0.686	<b>0.741</b>
London	852	0.393	0.302	0.775	0.559	0.558	<b>0.739</b>
Lisbon	327	0.386	0.336	0.529	0.525	0.581	<b>0.762</b>
Paris	1057	0.262	0.139	0.832	0.687	0.701	<b>0.752</b>
Dublin	230	0.381	0.183	0.730	0.607	0.679	<b>0.738</b>
Rome	721	0.421	0.381	0.619	0.512	0.571	<b>0.787</b>
Vienna	288	0.547	0.444	0.979	0.553	0.630	<b>0.818</b>
Prague	459	0.458	0.353	0.904	0.552	0.609	<b>0.794</b>
Copenhagen	153	0.309	0.118	0.601	0.628	0.625	<b>0.816</b>

the user in booking it. On the contrary, a high price in the low availability condition makes it appealing for the users to book it. The positive correlation of the highPrice with the probability of a hotel to be fully booked can be explained by the fact that in the low availability condition, the hotel with the higher prices are typically the best ones remained. In the high availability condition, instead, users tend to book cheaper hotels.

## 7.6 Discussion

We developed CDAHR, a method that outperformed the state of the art in the problem of predicting the hotel availability. This study has a lot of implications in the E-tourism domain. First of all our study pinpoints that considering only average rating and popularity is not enough to capture the user booking intentions. Considering context, instead, greatly increases the prediction performance. Thus the addition of context to the model helps in modeling the booking process of the user, that jointly considers the rating, the popularity, the price, the stars, the distance from the center and the general availability conditions of the city. This chapter shows that the context can be very valuable even in non personalized settings, such as the hotel booking domain. Besides identifying the factors that lead the decision making process in the hotel booking domain, we developed a high-accuracy context-driven recommendation method that can adapt to different occupancy levels. This can be useful in hotel booking domain sites, to warn the user that the hotel would not be available soon. It can also be helpful for balancing the short-head/long-tail distribution of hotels, by recommending users hotels that are in the long tail, possibly at lower fares.



**Table 7.2:** Weights learned for each city by the SVM classifier. Proximity is related to the proximity of the hotel to the city center. Occupancy is the percentage of booked hotels over the total number of hotels. Stars are the hotel stars, while lowPrice and highPrice are the prices for high and low season conditions, respectively.

City	rating	popularity	proximity	stars	highPrice	lowPrice	occupancy
Moscow	-0.090	0.213	-0.019	0.247	-0.992	0.678	-0.325
Athens	-0.090	0.215	0.000	0.265	-0.958	0.657	-0.332
Stockholm	-0.090	0.214	-0.010	0.265	-0.996	0.683	-0.324
Monaco	-0.083	0.205	-0.041	0.250	-0.896	0.607	-0.340
Amsterdam	-0.081	0.209	-0.018	0.260	-1.044	0.711	-0.320
London	-0.105	0.194	-0.021	0.255	-1.102	0.780	-0.338
Lisbon	-0.081	0.206	-0.020	0.247	-0.991	0.672	-0.331
Paris	-0.108	0.267	-0.007	0.278	-1.133	0.771	-0.256
Dublin	-0.085	0.221	-0.009	0.265	-1.003	0.674	-0.323
Roma	-0.082	0.210	-0.016	0.221	-0.960	0.654	-0.346
Vienna	-0.088	0.197	-0.024	0.258	-0.987	0.670	-0.306
Prague	-0.071	0.197	-0.028	0.251	-0.973	0.650	-0.315
Copenhagen	-0.092	0.212	-0.011	0.259	-0.996	0.682	-0.325



---

# CHAPTER 8

---

## Mobile Apps

---

In this chapter we explore the Mobile App Recommender Systems (MARS). We will introduce two different problems addressed in this field: the *application discovery*, which aims at suggesting apps to be installed on the smart device to the user and the *usage prediction* related to suggesting installed apps to be opened, for a quicker and handier launch. We will present a detailed literature review on these two problems but we will focus on the usage prediction because of its context-driven nature. We present our hybrid approach to the usage prediction that combines the Most Frequently Used and the Naïve Bayesian algorithms and improves the performance in terms of Mean Reciprocal Rank, Precision and Mean Average Precision. The superior performance of the proposed algorithm highlights the task oriented nature of the usage prediction problem and its dependency from the context.

### 8.1 Introduction

---

The mobile app market is growing at a tremendous rate. According to Gartner<sup>1</sup> the mobile app market will grow at least five times faster than the internal IT organizations' capacity to deliver them by the end of 2017. Gartner forecasts 2.1 billion mobile phones by 2019 which will fuel the demand for apps in the enterprises.

Mobile apps domain is the perfect example to describe this transition. The peculiarities of this domain depicted in Figure 1.1 are: *life cycle* and *task and intent*. The life cycle property that breaks the ImP paradigm is relative to the trendiness of the apps: many of them have a trendy period after which the user interest in this app decreases. The second peculiarity is relative to the strong task oriented nature of the mobile app domain. Usually users want to accomplish a goal with their mobile devices and, in

---

<sup>1</sup><http://www.gartner.com/newsroom/id/3076817>

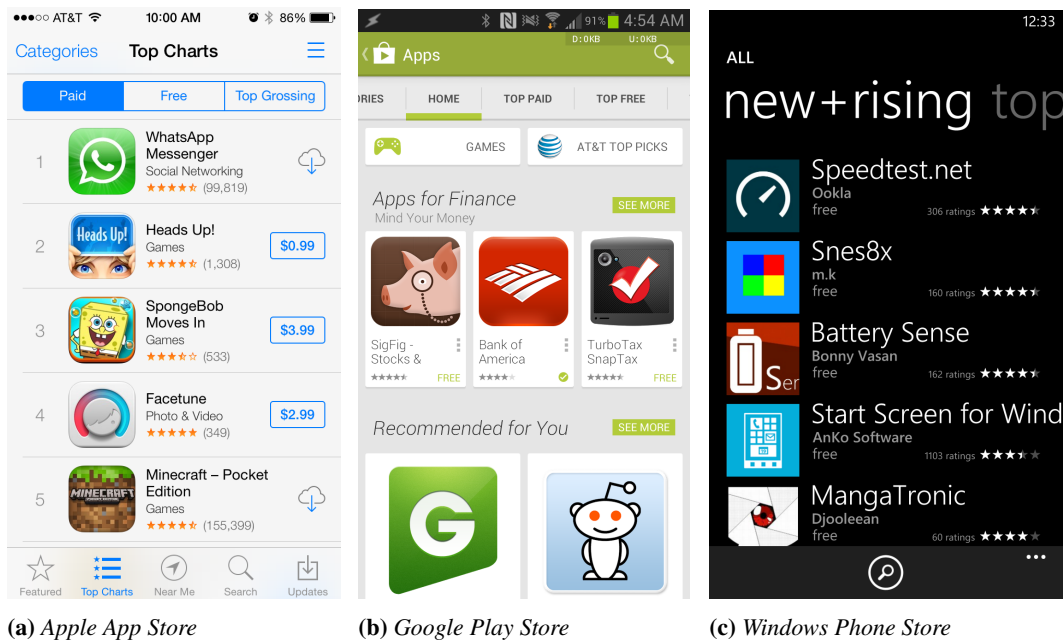
order to accomplish this goal, a number of applications have to be opened. The sequence of apps opened by a user can be seen as a session (or workflow) to perform a particular task. Suppose that a user wants to have dinner in a restaurant. First she may open instant messaging apps to agree with her friends for the schedule. Then she may open a browser or a review app to check the reviews of the restaurants in order to choose the one that best fit her needs. These needs can be strongly *context-driven*. For example she could have agreed to eat Japanese food, or to eat outside because the weather is lovely, or to eat near the workplace. Then she may open a navigation app in order to reach the place. This sequence of opened applications can be recurrent in other users' usage patterns. The process of finding and opening a new application can be optimized by suggesting in the home page the applications that the user is probably going to open next according to the system's understanding of the user's goal. The contextualization trend is felt both from academia and industry: recently Google has released the *context-awareness APIs* that allow any Android application to request high level, aggregated contextual information and to create triggers when the context changes<sup>2</sup>.

The process of installing a new mobile application is usually done via an app store. All of the app stores nowadays employ a recommendation algorithm to suggest new apps to the user on the basis of the users' past history of downloads, context and editorial needs. Figure 8.1 is showing the interface of the most common mobile app stores. Besides the official app stores there are plenty of unofficial ones which have their own interface and their own recommendation algorithms. However these app stores usually contain links to the official app stores. The only exception is Android that allows users to install applications from not official sources. In this way a custom app store can be a mobile application (or a website) from which users can directly download apps. Research literature has widely addressed the problem of *app discovery*, i.e. the recommendation of new apps to be installed by the user. Many online evaluations were using custom app stores on the Android platform [67, 133, 21, 143, 40, 11, 39, 130, 146, 117].

Each of the major mobile operating systems has its own method to present the list of apps to the user. Windows Phone adopts a tile interface, where users can rearrange the app icons and also change their size and appearance. Apple iOS has a paged interface where each installed app is present. Users can rearrange them and create folders. Android instead has a different approach: the home screen does not contain each installed application by default, but the user can access the list of installed applications and drag them into the home page. Although Android has no official default home screen, it allows third party applications (i.e. launchers) to customize the home page. There are plenty of launchers to choose from: some are showing mobile app recommendations (e.g. Yahoo Aviate Launcher) that allow the user to quickly open installed applications. Some other launchers are more classical, e.g. they allow the user to create as many pages as he wants and fill them with the application they want to access fast (e.g. Nova Launcher, Trebuchet). Google's standard launcher is Google Now Launcher which presents the users with a page containing its Google Now suggestions and updates and lets also users arrange the applications icons into other pages. Although Windows Phone has its own standard launcher, it allows users to download custom ones. Apple iOS, instead, does not allow to change the default launcher but has a page in which

---

<sup>2</sup><https://developers.google.com/awareness/>



**Figure 8.1:** Most widespread app stores

presents the most frequently used applications and contacts for quick access<sup>3</sup>.

## 8.2 Background

There are two main scenarios in which app recommenders can be employed: *installations* and *usage*. In the installation (or *app discovery scenario*) [139] the system recommends applications that are new to the users, e.g. the store recommendations. Usage prediction [70] is relative to the suggestion of the application to be launched. In principle, the same algorithm can be applied to both scenarios but different scenarios benefit from specialized algorithms. There is a third scenario that is relative to mobile app security issues such as malware detection. However we will not address these works because they fall outside of the scope of this chapter, addressing the problem from the security point of view.

### 8.2.1 Mobile application installation scenario

Finding the right application for the user need is very difficult. A report<sup>4</sup> states that nearly 3 out of 10 applications are instantly deleted after installation. This highlights the need for better recommender systems that understand the current need of the user. Most of the work on mobile app recommendation is relative to the installation scenario which is similar to the classical recommendation scenarios.

The recommendation of mobile apps to be installed poses two main challenges with respect to traditional recommender systems: size of catalog and negative feedback. The first challenge is relative to the tremendous number of applications available for download for app stores, which is around 2 million for both Apple App Store and Google

<sup>3</sup>as of 2016

<sup>4</sup><http://www.appbrain.com/stats/number-of-android-apps/>

Play store and around 600k for windows app store<sup>5</sup>. This huge catalog is challenging for the both scalability and accuracy of recommender systems. As a counter-example, traditional movie recommender systems are dealing with much smaller catalogues: the dataset relative from the Netflix Prize contained information on about 17k movies, while IMDB today has information about 355k movies<sup>6</sup>. The large number of items makes the dataset very sparse and this has also a huge impact on the performance: differently from the movie domain, it is more difficult to spot the right applications that the user will install and this usually lowers accuracy.

Another issue is relative to the feedback: in this setting a combination of explicit (e.g. the ratings) and implicit (e.g. the installation of an app) feedback is available. Moreover the event of installation of an app is a vague indicator of whether the user likes an application [70]. According to a recent survey<sup>7</sup>, almost a third of mobile applications are instantly deleted after installation.

Hereafter we list the papers in this direction.

**AppJoy: Personalized Mobile Application Discovery** [139] This is one of the first and most cited papers on mobile app recommendation. The authors develop Slope One, a recommendation algorithm that uses the usage data of applications with a CF technique, ignoring the user ratings, following the “rate with your feet” principle: what the user does matters more than her ratings when profiling her application needs. The authors perform an extensive online evaluation of their system.

**Climbing the App Wall: Enabling Mobile App Discovery through Context-Aware Recommendations** [67] Karatzoglou et al. develop a context-aware recommender system for a custom app store called *appazaar*. Their approach to the problem pivots around the tensor factorization, a context-aware recommendation algorithm, specifically the *Djinn Model*: they build a *user-item-context* tensor, where each context is a separate dimension. In this way they are able to model user tastes in different contexts, such as *moving*, *location*, *time of the day*, and *day of week*. As additional contextual variables they also use the number of times an app was used, thus leveraging usage data. Baltrunas et al. in [11] perform offline and online experiments to validate tensor factorization algorithm. They create a custom app store called Frappé where their algorithms are tested in a large scale environment. The dataset used for the paper was released to the community (See Section 8.2.3).

**Interoperability-Enriched App Recommendation** [133] Wenxuan and Airu describe a recommender system that exploits cross-store data to provide recommendations. Their approach is similar to [67], but they use the store as additional contextual variable. They employ a user-based similarity measure to recommend new apps to user. In their work they are able to track the same app in different stores. By using cross-store data, their model can learn more powerful and detailed relations between users and mobile apps.

---

<sup>5</sup><http://goo.gl/DFxBec>

<sup>6</sup><http://www.imdb.com/stats>

<sup>7</sup><http://www.appbrain.com/stats/number-of-android-apps/>

**AppTrends: A Graph-based Mobile App Recommendation System Using Usage History** [8] Bae et al. present a recommender system based on graphs and on app usage data to provide mobile app installation recommendation. They create a graph where the nodes are apps and the edges are relative to the co-occurrences of application usage of a user in a particular session. They define the session as the sequence of apps opened between the moment the user turns on the phone until the screen goes off. For recommendation they define a *fit score* that computes the similarity between the user graphs when adding a new node (app) to the existing graph.

**A Context-Aware Retrieval System for Mobile Applications** [89]. The authors of the paper develop a TF-IDF algorithm [19] which uses the description of app and the context of the user to infer user interest first on topics (categories) and then project it to apps. They carried out an online evaluation to test the effectiveness of their approach with 16 participants. They adopt a bag of word a approach to deal with context descriptions and app metadata.

**App Recommendation: A Contest between Satisfaction and Temptation** [142] Yin et al. model the process of adoption of an app as a *contest* between the owned app's actual value (i.e. how much the user likes an installed app) and the candidate app's tempting value, (e.g. how much the user is willing to install a new app). For example a user is less likely to download a weather forecast app if she already has one. They introduce the concept of *continous consumption*, in which the consumption of an app is not *one-shot*, such as with movies or books, but it serves the user continuously. The idea of modeling the app installation in this way is new to recommender systems and standard Collaborative Filtering techniques are not able to infer such kind of relations. It is important to point out that in a real scenario if the tempting value is high enough (even if not higher than the actual value of the existing app in contest) the user will download the new app, test it and then decide which app to keep, since many apps are uninstalled right after their installation.

**Evolving Mobile App Recommender Systems: An Incremental Multi-objective Approach** [136] Xia et al. adopt a multi-objective approach to optimize the recommender system towards ranking, diversity, revenue and robustness metrics. They propose a variant of the Latent Semantic Analysis (LSA) [48] algorithm to compute recommendations. Moreover they model the evolution of the system by introducing the method of *rank aggregation*, which is denoted as deriving a "consensus" ranking of the alternatives, given the diverse ranking preferences of various criteria. Their method outperform standard LSA method, but they did not compare the performance with standard Collaborative Filtering techniques.

**Personalized Mobile Application Discovery** [141] Yang et al. use the installed apps and their relative usage as input to their recommender system. They use installations to compute the popularity of apps and the usage to compute the user interest. In order to model the user interest, not-so-popular apps have much more descriptive power. As an example, a swimming app and a chat app may be used with the same frequency, but the swimming app reflects the user's interest much better than the generic chat

app. They apply a weighting scheme derived from TF-IDF and employ the Slope One algorithm [139].

**App Mining: Finding the Real Value of Mobile Applications** [143] Yu and Au Yeung use also uninstallation data as negative feedback. In their model, named “Actual Value”, the perceived value of each app is computed on the basis of the time elapsed between the installation and the uninstallation of that specific app and on user’s “expertise”.

**Swarm Intelligence methods** [137, 135] Xia et al. apply swarm intelligence to optimize an objective function chosen from similarity, diversity, and utility. Their model is inspired by the organization of bullets in a revolver gun, from which the algorithm takes its name, Cylinder Filling, since the solution is fixed size and has a ring structure. In this cylinder, the bullets are the recommendations. Their algorithm works by successive refinements to the “bullets” via particle swarm optimization, maximizing the objective function.

**Top-N Selection** [40, 39] Cui and Liang exploit the fact that user interests spawn across different categories to diversify the recommendation list. Their approach is to generate the recommendation list with a standard Collaborative Filtering algorithm and then balance the number of apps from different categories on the basis of the previous user history in these categories. The experiments have been carried out using two categories: *games* and *utilities*.

**A Novel APP Recommendation Method Based on SVD and Social Influence** [130] Wang et al. explore the addition of social influence to the field of app discovery. They use a Data-Based Influence Diffusion Model to model the reciprocal influence of users and embed it in an Singular Value Decomposition (SVD) [71] based recommendation algorithm that leverages usage data. They defined the social influence in a non-symmetrical way, thus allowing the discovery of “seed” users. Their evaluation has been conducted offline using Tencent App Store data of 10k users.

**A combination of temporal and general preferences for app recommendation** [65] Jang et al. use topic modeling to recommend cold start apps to users. They use a model to capture users’ temporal preference patterns and Latent Dirichlet Allocation (LDA) [19] to learn the general user preferences. They used three users to validate their hypothesis.

**What Special about Top-N Recommendation for Mobile App Stores** [146] Zhu et al. try to determine if browsing history, app update history, and app categories introduce a bias in user app downloads. Moreover they compare the performances of different recommendation algorithms in different app categories. As for the user browsing history, they found out that the last hour of browsing is directly related to the user app downloads. Therefore remote past history should be used to infer user interest, not as a way to infer app downloads. For app update, they found out that the action of updating an app is correlated with app usage. App categories are also correlated to user app download, with some user preferring certain categories. According to their study there is not an algorithm which is clearly better for all app categories, so the authors suggest to use different models for different categories.



**AppFunnel: A Framework for Usage-centric Evaluation of Recommender Systems that Suggest Mobile Applications** [21] The authors of the paper develop a method for a comprehensive evaluation of mobile app recommenders. They introduce different metrics following the life span of an application, from the *recommendation*, through *view in store*, *installation*, *direct usage*, and *long term usage*. They use the *appazaar*<sup>8</sup> app store to carry out the evaluation. Their main finding is that context-aware recommenders are better in the short term, while non-context-aware ones are better in the long term.

### 8.2.2 Mobile application usage scenario

This scenario is relative to “app launchers”, i.e. contextualized user interfaces for smartphones. Instead of presenting applications in static order, these app launchers use recommendation algorithms to highlight apps that a user will most likely need in the current context. An example of this is the *Yahoo Aviate Launcher*<sup>9</sup> for Android devices that presents the users with an application list tailored to their needs.

Differently from app installation setting, usage prediction is limited to the apps that are currently installed by the user. According to a survey<sup>10</sup> in 2014 the average number of installed apps on Android phones is 95. While this fact makes prediction easier because the predictions for only the installed apps have to be computed, the training of a collaborative model should be able to handle nearly all the item set, thus increasing the training complexity.

Moreover users tend to use applications much more often than installing them. This can be an issue for models that take into account the sequentiality (i.e. time series) of the opened apps .

The peculiarity of this scenario is its strong context dependence. The authors of [22] argue that context is the main driver for mobile app recommendations. One of their findings is that the app usage is maximum in afternoon and evening, peaking around 6PM. According to this study the communications app are used every hour of the day. News apps are likely to be used in the morning. After communication apps wind down during the evening, games become predominant. Social apps have their highest probability of usage in the late evening (from 9PM to 1AM). The authors also defined the concept of *application chain* i.e. a sequence of apps opened without the device being in standby mode for longer than 30 seconds. They found that 68.2% of the applications chains are composed by a single application, 19.5% by two and 6.6% by three. According to this finding the vast majority of users tend to use a very small set of apps when interacting with their smart device. However [8] and [63] reported opposite results with respect to the importance of the session. The authors of [22] also found empirical evidence for location as a co-variate for mobile app usage behavior. For example they found differences between Europe and US, or by examining functional regions like inside or outside the airport. An interesting finding is that when moving faster than 25km/h users tend to use multimedia applications and, surprisingly, they are less interested in travel applications. With this study authors confirm that time and

<sup>8</sup><http://matthiasboehmer.de/appazaar/>

<sup>9</sup><http://aviate.yahoo.com/>

<sup>10</sup><http://thenextweb.com/apps/2014/08/26/android%2Dusers%2Daverage%2D95%2Dapps%2Dinstalled%2Dphones%2Daccording%2DYahoo%2DAviate%2Ddata/>

location are the most important contextual signals that drive the mobile application recommendation.

In the following we list the research papers in this area:

**Preference, Context and Communities: A Multi-faceted Approach to Predicting Smartphone App Usage Patterns** [138] Xu et al. develop a bag of features approach to model the contextual features and a similarity metric to perform recommendations. Their approach leverages three key factors that affect everyday app usage decisions: (i) intrinsic user app preferences and user historical patterns, (ii) user activities and the environment as observed through sensor-based contextual signals, and (iii) the shared aggregate patterns of app behavior that appear in various user communities. They test their method with offline data, involving 4606 users and with an online experiment involving 35 users.

**Contextual Adaptive User Interface For Android Devices** [64] Jain et al. develop a contextual adaptive user interface not only for recommending apps, but also for some action buttons (e.g., call or text this contact). Their interface is reactive to device sensors readings, user location, duration of contact, category of person being contacted, day and time, and application usage logs. The interface adaptations involve changing ordering of menu items, prominently display UI predicted elements, selectively displaying application icons, changing the positioning of icons, and overlaying predicted icons on current application screen. They test their custom interface with 4 users.

**Mining Mobile Application Sequential Patterns for Usage Prediction** [83] Lu et al. use the *Mobile Application Sequential Pattern Mine* algorithm to model the sequence of opened applications. Their location-based approach first discovers the stay locations for the user and then mine the sequence of applications for these places as well as for the paths between these locations. Their objective is to predict the usage of applications to preload them in memory for a faster app launch. Offline experiments validate their model against most-recently-used and most-frequently-used baselines.

**Conditional Log-linear Models for Mobile Application Usage Prediction** [70] Kim and Mielikäinen use Conditional Log-linear Models to predict application usage. The model describes the conditional probability of application usage given observed context variables. The approach is based on discriminative models, which uses logistic regression and conditional random fields. The contextual variables used are time, location, wifi network and last used apps. They evaluate the algorithm offline using data from 142 users with at least 2,500 usage events each.

**Prophet: What App You Wish to Use Next** [148] Zou et al. predict app open events with Bayesian models. They compare their method with existing baselines, such as Most Recently and Most Frequently used. Bayesian models slightly increase the performance with respect to baselines.

**Predicting The Next App That You Are Going To Use** [9] Baeza-Yates et al. use a variety of features representing the real-time spatio-temporal contexts and the session to predict

mobile app usage using Tree Augmented Naïve Bayes. They conduct an extensive offline evaluation using Yahoo Aviate Data. The authors also propose some methods for addressing both the user and app cold start problem. For the app cold start they divide the apps in two categories: short and long term apps. For both categories they exploit the concept of “wisdom of the crowd”, i.e. other user’s usage pattern of the same app. The difference is that for short term apps, after some hours the estimations can be replaced by the actual usage of the user, while for long term apps the replacement happens more smoothly. For cold start users, they propose two strategies: (i) provide the recommendations from most similar users on the basis of the catalog of installed apps and (ii) create a fictitious usage history profile on the basis of the similarity of the catalog of the installed apps (the pseudo-user strategy).

**On the Feature Discovery for App Usage Prediction in Smartphones** [75] Liao et al. use the Minimum Description Length criterion to select features used for the kNN app usage prediction. They model two different kinds of features: the Explicit Feature (EF) from the readings of built-in sensors, and the Implicit Feature (IF) from app usage relations. The IF feature is derived by constructing an app usage graph that models app usage transitions. The graph is used to discover usage relations among apps. Since users may have different usage behaviors on their smartphones, they propose a personalized feature selection algorithm: minimum description length. It selects those features whose combination has the shortest length to describe the training data. They evaluate the algorithm with an online experiment involving 50 users against Most Recently Used (MRU), Most Frequently Used (MFU), Adaboost, Decision Tree and Bayesian methods.

**Understanding and Prediction of Mobile Application Usage for Smart Phones** [119] Shin et al. develop a personalized contextual launcher for Android devices based on Naïve Bayes method, after performing a study on the most useful contextual variables. The personalized interface aims at presenting the user with a app recommendations in order to reduce the cognitive effort in finding the app to be launched. An interesting point is raised in the discussion of the paper: users may be disoriented if the order of apps is continuously changing, thus mitigating the effect of the algorithm in reducing the cognitive effort. The authors suggest to have a static part in the home screen with most frequently used apps and a dynamic part, containing the other recommendations.

**Practical Prediction and Prefetch for Faster Access to Applications on Mobile phones** [102] Parate et al. use a method based on the prediction by partial match [90] to predict the usage of apps and also to prefetch the apps in memory for faster app launch. Prediction by Partial Match is one of the widely used methods in text compression. At a high level, PPM operates by scanning character sequences, and building up a variable-length Markov-based predictor on the fly. However for PPM the longest prefix is the most relevant, while for apps both long and short sequences are relevant for recommendations. The paper also describes methods for estimating which is the *right time* to prefetch an application in memory. In order to test the effectiveness of their approach, they developed a widget for the Android platform, which has been installed 43k times and has 7,630 active users.

**Fast App Launching for Mobile Devices Using Predictive User Context** [140] Yan et al. describe FALCON, a method based on contextual variables, such as user location and temporal access patterns, to predict app launches before they occur. This algorithm tries to model sessions (tasks) of users with the idea of triggers and followers: trigger events, for example answering a phone call or reading an text message, have a tendency to open a sequence of other applications (followers) within a relatively short time interval between each other. Their algorithm relies on temporal bursts: some users may use an app, e.g. a game, continuously for a period of time and then do not open it for several months. The combination of these insights and the spatial and temporal contextual variables make the algorithm effective in predicting app usage. They prototyped FALCON on a Windows Phone with minimal impact on the system energy consumption and memory footprint.

**MobileMiner: Mining Your Frequent Patterns on Your Phone** [122] Srinivasan et al. use sequential pattern mining and temporal patterns to address the app launch problem. The algorithm mines frequent co-occurrence usage patterns directly on the device and derives association rules among apps and between apps and context, such as {Morning}  $\rightarrow$  {UseGmail}. They evaluate their approach online by developing an Android application that has been installed by 106 users in a period of 3 months.

**On Mining Mobile Apps Usage Behavior for Predicting Apps Usage in Smartphones** [76] Liao et al. used temporal and periodical features to improve top-k recommendation. On the basis of the explored features, the algorithm dynamically derive an *app usage probability model* to estimate the current usage probability of each app in each feature. They propose two selection algorithms and evaluate the performance by using two real mobile app usage traces.

Algorithms that are employed in this scenario are very different to each other and try to model a different contextual aspect. This diversity of approaches reflects the novelty of this domain and the lack of a well-established solution. This is deeply connected to the context modeling: the presence of context and its importance in this field broadens the category of algorithms to be used, spanning from information retrieval techniques, to data mining, optimization, and even text-compression techniques, showing that the transition from personalization to contextualization is not easy nor straightforward.

### 8.2.3 Datasets

The field of mobile application recommendations is not very well tackled by the recommender systems literature. This is partly due to the lack of publicly available datasets, as mobile application companies do not disclose their findings in this field. Most works in this area exploit private datasets. The only publicly available datasets are Frappé<sup>11</sup> and Nokia Mobile Data Challenge (MDC)<sup>12</sup> (see Table 8.1).

Many studies on installation leverage both installation and usage data. In fact, while the feedback of an installation is not very informative (the application can be uninstalled right after its installation or never be used), the usage data is indicative of user's interests

---

<sup>11</sup><http://baltrunas.info/research-menu/frappe>

<sup>12</sup><https://www.idiap.ch/dataset/mdc>

and habits. In the usage scenario, on the contrary, only usage data are used but they are linked with contextual information. Besides time and location, which are the most common contextual signals used, many studies use other contextual variables such as battery level [122], wifi SSID [70], speed of connection [145], accelerometers [138], the presence or absence of headphones [9], and other sensory data [63].

All datasets used in the literature come from either online app stores or from monitoring apps installed on the phone. Datasets from app stores can be collected either by crawling online stores (e.g. Google Play Store) or by creating custom app stores. Datasets collected directly from apps running on smartphones contain installation, uninstallation and app usage profiles. Some apps perform the monitoring activity as a side effect for their main task (e.g. app launchers). Others apps are designed with the main purpose of profiling users.

**Crawl** Crawling from official app stores is the easiest way to obtain mobile app data. Datasets crawled from existing app stores do not contain user’s signals (e.g. installation, uninstallation) that are used to train CF algorithms and, more importantly, as ground truth to evaluate RSs. These datasets can be used only to build CBF algorithms allowing to gather information only on app category, app description, average ratings and some reviews. This is why papers using crawled data [89, 137, 135] describe non personalized algorithms, that mainly work with app description and categories and do not require data about user activities.

**Custom App Store** Some works use data coming from custom app stores. A custom app store is a service that presents the user with apps to be installed on the device. It can be a *separate* store, which physically hosts the apps and provides the possibility to download them directly (available only on the Android platform, e.g. Amazon App Store) or it can be a *proxy* app store, which points to the original app store in the moment of installation. An example of *separate* app store is the *appazaar* app store<sup>13</sup>, an academic project (now discontinued) that provided data for both online and offline testing to: [67, 20, 21, 22, 23]. An example of a *proxy* custom app store, available for the Windows Phone platform, is Wpapps, used in [133]. The advantage of the app store with respect to crawling is that it can log the browsing behaviour of the users, e.g. click on an app or read its description. However, differently from the separate custom app store, a proxy app store cannot directly log the event of installation. If a custom app store is provided as a phone app, it can leverage the data coming from both the information coming from the store and the information gathered as an actual app installed on the phone.

**App collecting data** Any application installed on the phone has the possibility to gather data about the installed apps and their usage. Different apps have been developed for this purpose: AppSensor application is used in [22], AntTest in [145], AppTrends in [8], AppJoy in [139], Device Analyzer in [54], and AppKicker in [102]. Other apps for which the name is not provided are used in [119, 76]. [140] gave an iPhone for 14 months to get usage and contextual data data.

<sup>13</sup><http://matthiasboehmer.de/appazaar/>

**App Launchers** An app launcher is an application that manages the home screen of a smartphone. Examples of app launchers are Google Now Launcher, Nova Launcher or Yahoo Aviate Launcher, all of them for the Android platform. Being the app launcher an actual application installed on the device, it can collect data about the installed applications and their usage. The dataset used in [9] has been collected using Yahoo Aviate Launcher<sup>14</sup> for Android. Another launcher, iLauncher, is used in [123]. The advantage of the launcher over a normal app installed on the phone is that a launcher can log every event occurring on the home screen of the device, e.g. swipes over the different home pages.

The most used datasets in MARSs are *Nokia Mobile Data Challenge*, *AppJoy*, *NetSense*, and *Frappe*. [63, 70, 148] used the *Nokia Mobile Data Challenge 2012*, . Nokia Research Center Lausanne collected data from smartphones of almost 200 participants over more than year and released it for the research community. The dataset is be available upon request<sup>12</sup>. Another dataset, *AppJoy* was collected by the authors of [139] and subsequently used in [138, 141]. The dataset should be available<sup>15</sup> but the download link is not working at the time of writing. The *NetSense* dataset is related to a study launched in August of 2011 consisting of 200 incoming freshmen at the University of Notre Dame and was used in [86]. *Frappe*, was created for research in [11] and it is publicly available<sup>11</sup>.

Table 8.1 summarizes the characteristics of the datasets for installation and usage scenarios. The `#source` column refers to the source of data, either custom app store, app or launcher. Column `#events` refers to the total number of events in the dataset, either usage or installation or both. The type of events can be inferred by looking at `usage` and `installations` columns. Column `context` is marked if time and location of each event are present. Crawled datasets are not reported since this table does not contain any information about user activities.

### 8.3 Context-driven hybrid usage prediction algorithm

---

Our Context-Driven Hybrid Usage Prediction (CDHUP) algorithm is a combination of the two most common and effective usage prediction algorithms, namely Most Frequently Used (MFU) and Naïve Bayes. MFU is a simple method which recommends the most frequently used apps for each user. A Bayesian network is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph. In this graph the nodes represent the variables and the edges represent the probabilities. A Naïve Bayes classifier is a simple model that describes particular class of Bayesian network - where all of the features are class-conditionally independent. Our Naïve Bayes classifier computes the probability of opening each app conditioned to the context in which we want to make the prediction. The prediction can involve one or more contextual variables. Here is an example for the mathematical formulation of the probability with only one contextual variable:

$$P(i|C_1) = P(i, C_1)/P(C_1) = P(C_1|i)P(i)/P(C_1) \quad (8.1)$$

where  $i$  is the app for which we are calculating the probability and  $C_1$  is the con-

---

<sup>14</sup><http://aviate.yahoo.com/>

<sup>15</sup><http://www.cs.uml.edu/~glchen/sam/>

### 8.3. Context-driven hybrid usage prediction algorithm

Publication	Source	users	items	time span	events	usage	installations	context: time and location	available
[139]	app	4,600	16,950	400 d	10M	x	x		
[8]	app	206		2 w	4,2 M	x			
[140] [141] [138]	app	1,325	1,489	14 m	1.3 M	x	x	x	
[22]	app	4,125	22,626	163 d	4,92 M	x		x	
[145]	app	13,969	16,878	485 d	103 M	x		x	
[63] [70] [148]	app	200		≥ 1 y	1M	x		x	x
[64]	app	≥ 4	50			x		x	
[83]	app	30	> 100	5 m	3,432 seqs	x		x	
[86]	app	200		3 y		x			
[54]	app	30			1.5M	x			
[123]	app	34		98 d		x			
[75]	app	50				x		x	
[119]	app	23		1 m		x		x	
[102]	app	100		14 m	17k	x		x	
[122]	app	106				x		x	
[76]	app	12	636	5 m	42k	x			
	app	80	3,280	15 m	1M	x			
[67]	store	3,260	18,205		3,7 M	x	x	x	
[133]	store	30k	1M		10M	x	x		
[117]	store	100k	7k	2 w	1.82M	x	x		
[142]	store	3,308	5,661				x		
[143]	store	460k	62k	1 w			x		
[11]	store	1,000		2 m	351k	x	x	x	x
[39]	store	1,463	7,836				x		
[130]	store	300k				x	x		
[146]	store	220k	6,522	13 d			x		
[21]	store	45		3 m	287	x	x	x	
[9]	launcher	480		6 m		x		x	

**Table 8.1:** *Datasets*

sidered context. The following equation describes the mathematical formulation of the probability with 3 contextual variables involved:

$$P(i|C_1, C_2, C_3) = P(i, C_1, C_2, C_3)/P(C_1, C_2, C_3) = P(C_1, C_2, C_3|i)P(i)/P(C_1, C_2, C_3) \quad (8.2)$$

For the prediction our algorithm computes the probabilities for each app to be opened under one(or more) specific context(s). Apps with the highest opening probability will

be recommended to the user.

We used a maximum of 4 contextual variables:

- CTOD: time of the day. This variable divides a 24 hour day into four different intervals: morning (between 6AM and 12AM), afternoon (between 12AM and 6PM), evening (between 6PM and 12PM), and night (between 12PM and 6AM).
- CLOC: location. In the raw data, the longitude and latitude of the user are given for each type of event. Firstly, locations are clustered with the information that people mostly spend their time at work in the morning and the afternoon time of day and they tend to spend the night at home. After clustering, the places that the user has been the most are assigned as *work* or *home*. If a location is above 10 meters from home or work it is marked as *other*.
- CMOV: moving. When the user opens several applications within a short period of time, it is possible to observe if the user is stationary, walking or even in a vehicle moving faster. To classify the situation of the user in terms of movement, the movement speed is calculated. And they are grouped by the speed ( $v$ ) in meters per second (m/s) as follows: *Not available* if there are no consecutive events within 3 minutes, *Stationary* if  $0m/s \leq v < 0.1m/s$ , *Walking* if  $0.1m/s \leq v < 2.4m/s$ , *Faster* if  $v \geq 2.4m/s$  [67].
- CDOW: day of week. It represents the ordinal day of week, from Monday (0) to Sunday (6).

Our CDHUP is an ensemble of 5 different algorithms: MFU, Naïve Bayes with CTOD, Naïve Bayes with CLOC, Naïve Bayes with CMOV, and Naïve Bayes with CDOW. In order to merge the algorithms we are simply averaging the probabilities calculated by each algorithm for each app. Our algorithm is executed per user, i.e. there is no collaborative filtering involved.

## 8.4 Experiment setup

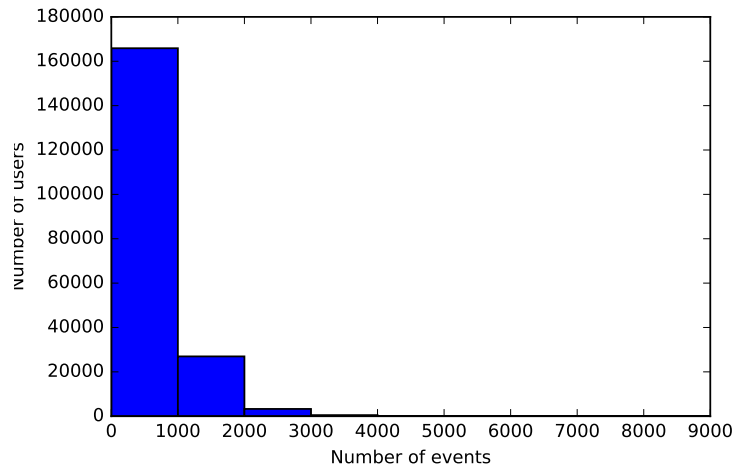
---

### 8.4.1 Dataset

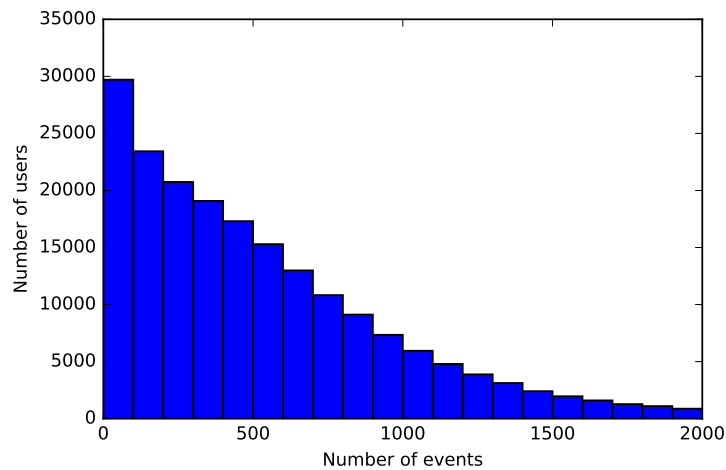
We used a dump of one week of 2015 of the Yahoo Aviate Launcher. The data contains installations and open events for any app the user installed. The raw data includes following information:

- User id
- App id
- Timestamp
- Latitude
- Longitude
- City
- Day of week  $\in \{0, 1, 2, \dots, 6\}$





**Figure 8.2:** *Distribution of app open events per user*



**Figure 8.3:** *Distribution of app open events per user limited to users having less than 2,000 events*

- Hour of day  $\in \{0, 1, \dots, 23\}$
- Event type  $\in \{ \text{open, install, uninstall} \}$

The number of installation events is 785,965 while the number of open events is 110,185,157 for a total of 110,971,122 events. The dataset also contains the list of apps installed by each user. The number of users is 201,513, the number of apps is 115,443 for an average of 33 apps installed by each user. Figure 8.2 shows the histogram of app open events for users. The x axis reports the number of events in the bin, while the y axis reports the number of users for that bin. As we can see the number of open events per user varies up to 8,500, but most of the users have less than 1,000 events. Figure 8.3 details the previous histogram for users having less than 2,000 open events. As we can notice the majority of users have less than 100 open events. Figure 8.4 shows the distribution of open events for the top 20 most used apps. The y axis report

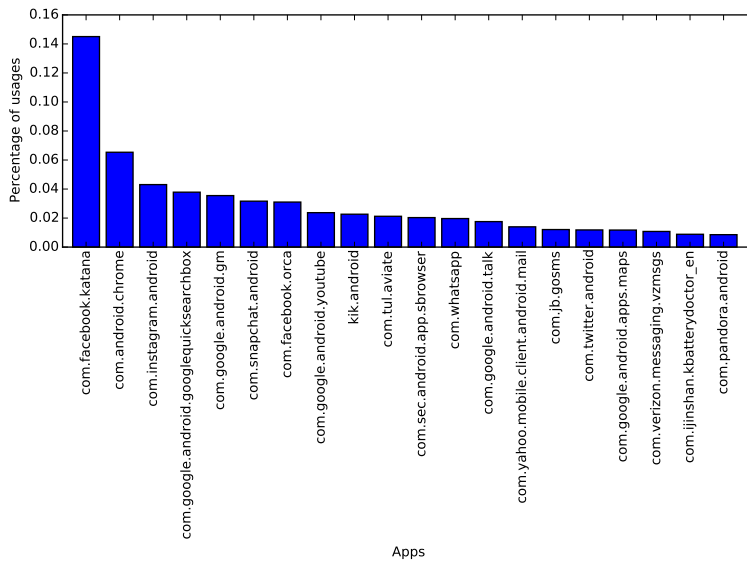
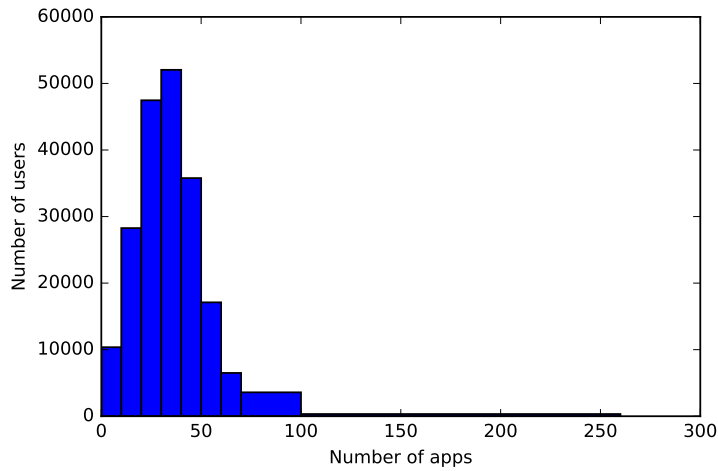


Figure 8.4: Distribution of app open events per app

the percentage of open events for each app. The x axis report the app ids, sorted by decreasing number of open events. This plot shows that top 10 applications account for 45% of all open events. Figure 8.5 presents the number of apps installed in the users’ devices. The x axis reports the bins for the number of apps, while the y axis reports the number of users in the bin. As we can see from the plot, most of the users have between 30 and 50 applications installed.

40.000 applications in our dataset is not present in the Play Store anymore as of November 2016. We included these apps in the UNKNOWN category. The reason for this is that some of these apps are native like call, contacts or settings, and some of them are apps provided by the manufacturer. Figure 8.6 shows the app category distribution before removing the apps not in the store anymore (8.6a) and after (8.6b) over the apps in our dataset. Categories are reported in the x axis and the number of apps in the y axis. We can see the UNKNOWN category is accounting for most of the apps while games becomes the most common category for the apps in the dataset after the filtering. Figure 8.7 shows the number of open events for the top 15 used application in the 3 location contexts considered. The x axis reports the name of the application while the y axis reports the number of open events for that application. Three consecutive colored bars represent number of open events when the user located at work, home and other. We can immediately notice that some applications are used in specific locations. Figure 8.8 shows the number of open events for the app categories in the location contexts considered. Some categories, such as FINANCE are mostly used at work. Figure 8.9 shows the number of open events during the time of day by the top 15 used applications. The x axis reports the name of the application while the y axis reports the number of open events for that application. Four consecutive colored bars represent number of open events during the day. We can notice that some applications are mostly opened during the evening. Figure 8.10 shows the number of open events during the time of day by the top 91-100 used applications. In this case it is even clearer that some apps are mostly used during particular times of the day. Figure 8.11 shows



**Figure 8.5:** *Number of apps installed in user devices*

the number of open events by time of day for the app categories. Some categories, such as MEDICAL, are mostly opened in specific times of the day.

#### 8.4.2 Evaluation Methodology

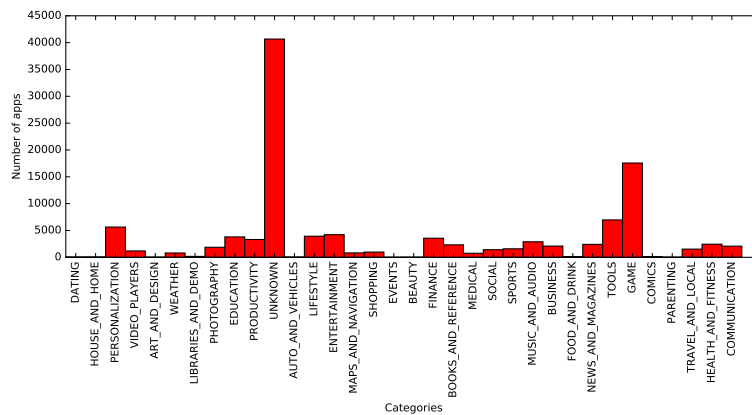
In order to evaluate our algorithm we split the dataset temporally. For each user we consider the first 80% of events as training set and the last 20% as test set. We used Mean Reciprocal Rank (MRR), Precision (PRE), and Mean Average Precision (MAP) metrics. For precision we used only the next event as ground truth, while for the other metrics we consider all the events in the test set as ground truth.

## 8.5 Results

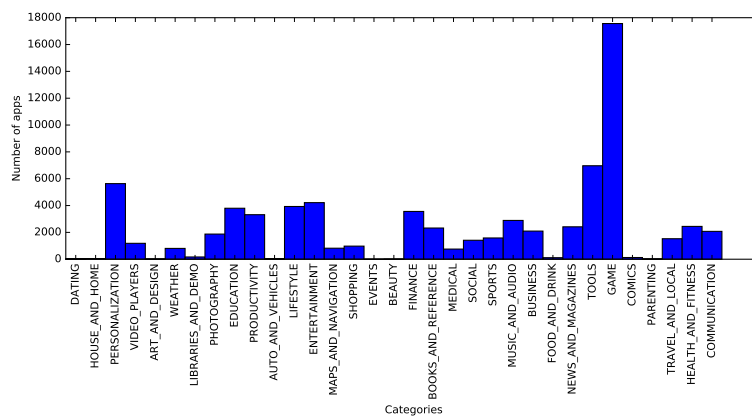
Table 8.2 shows the performance of the compared algorithm with four different dataset filtering conditions: only keeping the apps in store and keeping the apps in store but removing the top 10, 25 and 100 most used apps. The table compares MFU, Naïve Bayes (CTOD, CLOC, CMOV) an ensemble of three single Naïve Bayes predictors, each one only using one contextual variable, and CDHUP two different versions of our algorithm ensembling MFU with three and four single Naïve Bayes predictors each one using only one contextual variable. We can see that our CDHUP with 5 algorithms is outperforming the other algorithms with any filtering condition and with any of the considered metrics. Results show that removing the most used apps leads to performance increments. This indicates that our algorithm is better at predicting long tail apps than most popular apps. However the percentage improvement of our algorithm over the baselines is very similar regardless of how many top popular apps are removed.

## 8.6 Discussion

In this chapter we presented our work in the Mobile Application Recommender Systems field. We reviewed the state-of-the-art publication in the field and the datasets



(a) Distribution of app categories before filtering



(b) Distribution of app categories after filtering

Figure 8.6: Distribution of app categories before and after filtering

used, highlighting the challenges and the differences with respect to traditional recommender systems domains. We proposed an usage prediction algorithm able to outperform the baseline techniques. We also validated the ensemble theory, which says that the combination of weak learners produces a stronger learner. In the *app discovery* scenario the algorithms are closer to standard RSs, but many of them take into account different kind of feedbacks (installation and usage data). In the *app usage* scenario, instead, the problem is much different from traditional recommendation setting and context is crucial to this scenario. In this scenario many different approaches are developed, from sequential pattern mining, to swarm intelligence methods, through bayesian and tensor based techniques. By considering these two different problems we showed how prominently context assumed a central role in the user modeling. We also showed that taking into account the contextual information in the model improves the prediction accuracy.

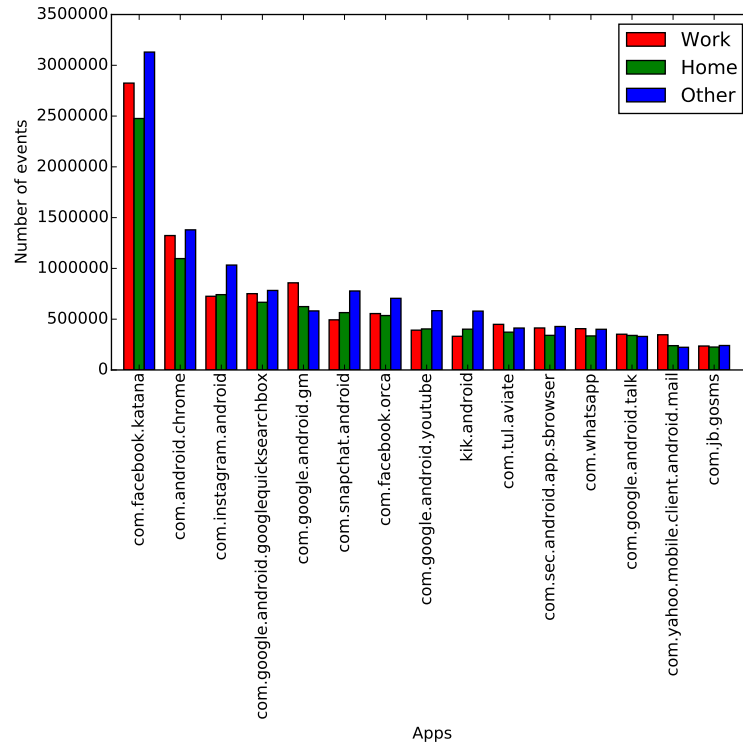


Figure 8.7: Open events in home, work, or other location for the top 15 used applications

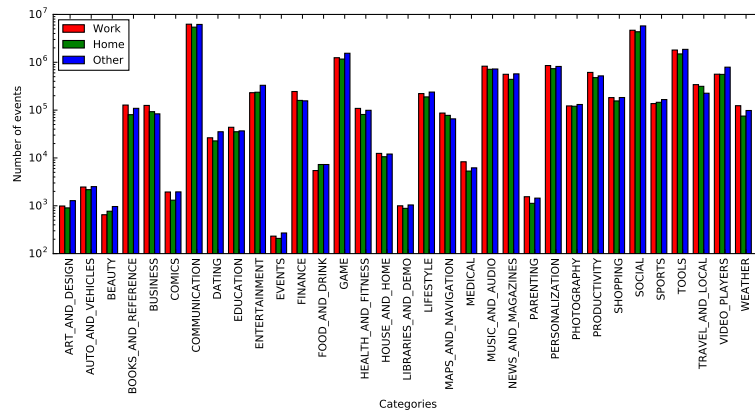


Figure 8.8: Open events in home, work, or other location for the app categories

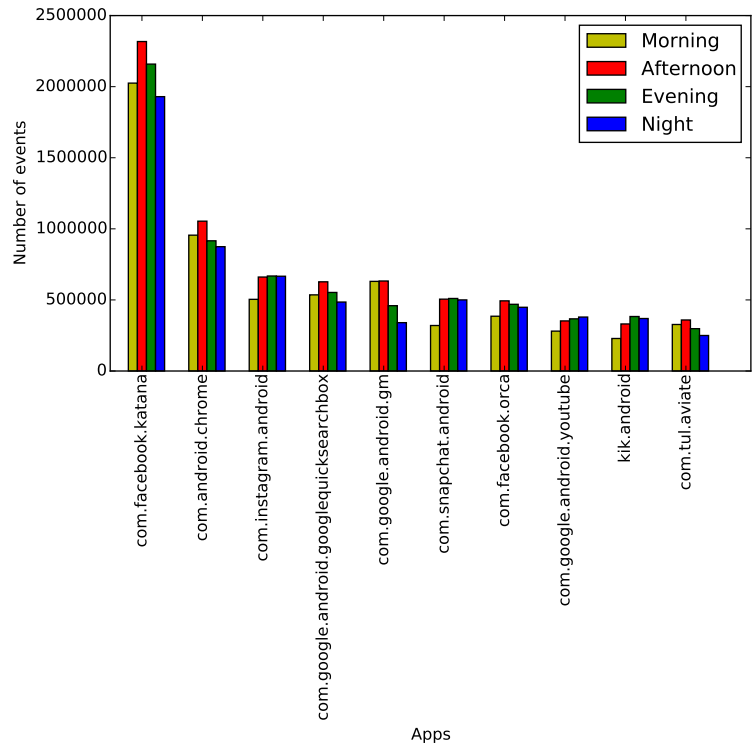


Figure 8.9: Open events during the time of day for the top 15 used applications

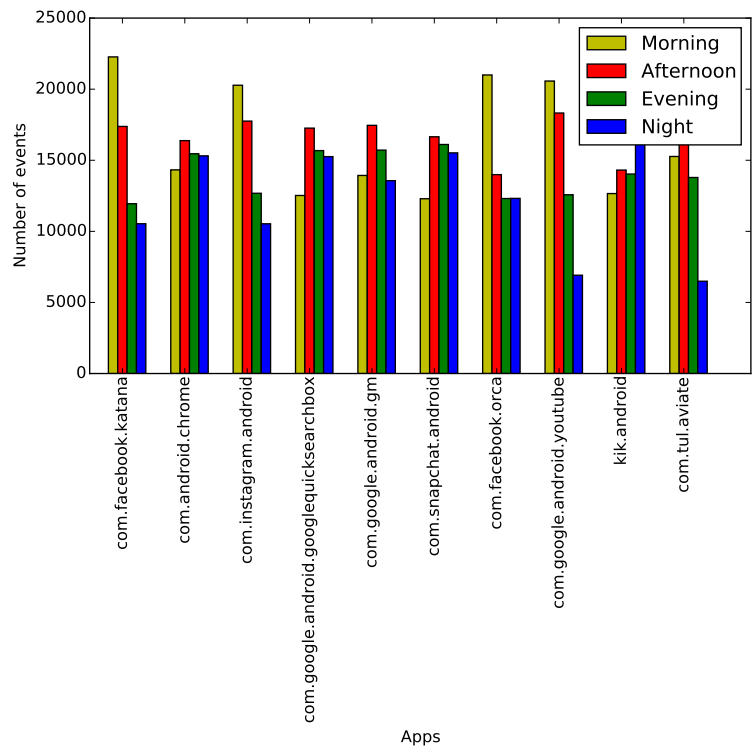


Figure 8.10: Open events during the time of day for the top 91-100 used applications

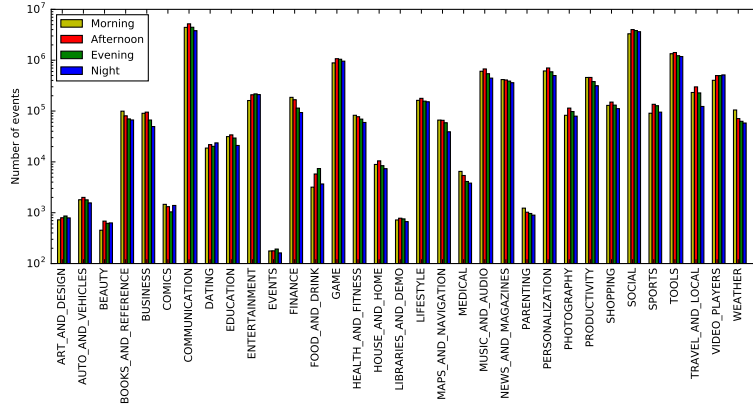


Figure 8.11: Open events during the time of day for the app categories

Table 8.2: Algorithms performance in terms of Mean Reciprocal Rank (MRR), Precision (PRE), and Mean Average Precision (MAP) with different dataset filtering conditions.

Filtering	Algorithm	MRR	PRE	MAP
in store filtering	MFU	0.4497	0.7019	0.8376
	Naïve Bayes (CTOD + CLOC + CMOV)	0.4588	0.7154	0.8377
	CDHUP (MFU + CTOD + CLOC + CMOV)	0.4594	0.7159	0.8408
	CDHUP (MFU + CTOD + CLOC + CMOV + CDOW)	<b>0.4526</b>	<b>0.7319</b>	<b>0.8459</b>
in store filtering no top 10 apps	MFU	0.4719	0.6942	0.7564
	Naïve Bayes (CTOD + CLOC + CMOV)	0.4894	0.7116	0.7657
	CDHUP (MFU + CTOD + CLOC + CMOV)	0.4902	0.7119	0.7657
	CDHUP (MFU + CTOD + CLOC + CMOV + CDOW)	<b>0.5058</b>	<b>0.7267</b>	<b>0.7747</b>
in store filtering no top 25 apps	MFU	0.4438	0.6689	0.6983
	Naïve Bayes (CTOD + CLOC + CMOV)	0.4477	0.6854	0.7033
	CDHUP (MFU + CTOD + CLOC + CMOV)	0.4516	0.6862	0.7068
	CDHUP (MFU + CTOD + CLOC + CMOV + CDOW)	<b>0.4598</b>	<b>0.7010</b>	<b>0.7172</b>
in store filtering no top 100 apps	MFU	0.4394	0.6544	0.6277
	Naïve Bayes (CTOD + CLOC + CMOV)	0.4483	0.6708	0.6386
	CDHUP (MFU + CTOD + CLOC + CMOV)	0.4502	0.6712	0.6401
	CDHUP (CTOD + CLOC + CMOV + CDOW)	<b>0.4631</b>	<b>0.6850</b>	<b>0.6566</b>





---

## CHAPTER 9

---

### Discussion and Conclusions

---

With this thesis we showed how in many domains *context-driven* recommendation can solve the problems of traditional recommendation methods, as depicted in Figure 1.1. Task and intent can be captured by looking at all the contextual signals, especially the session. The current session tells us which kind of items the user is looking for (in case of webshops) or can give a hint on the type of task the user is performing (Chapter 3). An ephemeral catalog can be addressed by estimating the preference of the user in features, instead of items (Chapter 5). We solved the seasonality problem in linear TV by creating a model for each time slot, eventually smoothing and blending them to perform recommendation (Chapter 6). The life cycle of the item can be addressed by modeling the usual item popularity curve in the algorithm itself, thus enabling it to suggest always new, fresh and trendy items (Chapter 4). The bounded capacity problem causes the best items, such as hotel or flight or restaurant, to be the first to become unavailable. In such a scenario, a *context-driven* recommender system should take into account the motivations that led the user to click on that particular item: was it because she likes that item regardless of the availability of other items or was it a choice over a limited set of items? We showed that a context-driven algorithm provided with such information (Chapter 7) is able to outperform state of the art techniques. We then aim at continuing our work on mobile app recommendation, where context is crucial to address the task/intent and the life-cycle challenges that this domain poses (Chapter 8).

Unfortunately we were not able to produce an answer to our research question which is valid in all of the considered domains, because each domain is peculiar and the importance of the contextual variables varies from domain to domain. Time and session, however, can be identified as the most useful contextual variables. Nonetheless the methodology for determining the best way to leverage context in each domain is similar: it is a mixture of common sense reasoning, availability of contextual variables

and empirical evaluation. For each domain we can have an hypothesis on the contextual variables that drive the item consumption process. Then this hypothesis has to be matched to the availability of the data we can use. Finally the utility of the contextual variables is evaluated by creating an algorithm that makes use of it and evaluating its performance. We presented many cases in which this methodology led to performance increments.

In many domains we found out that habits drive the item consumption process. On one hand, habits can be best leveraged by a personalized algorithm: an algorithm which can learn the user habits can better predict the next consumed items. On the other hand (i) habits of different users can be very similar, (ii) a user can have more than one habit, and (iii) habits occur in specific contexts. A context-driven algorithm does not try to model the user as a whole, but it tries to model the user as a collection of habits, each relative to a particular situation and intent. Actually a context-driven algorithm tries to model the habit itself and looks for occurrences of this habit in different contexts and different users. In other words, while from a personalized perspective habits can be seen as the expression of the user taste, from a context-driven perspective they are the expression of the tastes of many users when they are in the same situation and want to accomplish the same goal.

We pointed out the importance and the benefits of recommendations leveraging context. Now more than before it is time to focus on *context-driven* recommendations: there is the necessity, coming from new EU regulations that have been adopted as of April 14th 2016, and many enabling factors.

These new regulations give individuals more control over their personal data and make certain data less available to recommender systems, including:

- The need for the individual's clear consent to the processing of personal data
- Easier access by the subject to his or her personal data
- The rights to rectification, to erasure and "to be forgotten"
- The right to object, including to the use of personal data for the purposes of "profiling"
- The right to data portability from one service provider to another

They also lay down the obligation for controllers (those who are responsible for the processing of data) to provide transparent and easily accessible information to data subjects on the processing of their data<sup>1</sup>. These new regulations pose some challenges to the current state-of-the-art recommender systems. Basically every user can deny the use of its personal data. However, a *context-driven* recommender system is able to perform recommendations even without any personal information, using data in an anonymized way.

The most important enabling factor for *context-driven* recommender systems is the rising importance of the context due to the massive usage of smartphone and connected devices in our everyday life. Fitness bands, as an example, can measure our footsteps and our heartbeat; an ever increasing number of cars have smartphone-like operating

---

<sup>1</sup><http://www.consilium.europa.eu/en/policies/data-protection-reform/data-protection-regulation/>

---

systems, e.g. Android Auto; some products have started to control our houses, e.g. Amazon Echo; the TV itself is connected to the internet, opening possibilities that were not possible in the early Linear TV days. Bluetooth beacons and NFC communication have started to spread out, opening up for possibilities like location based or proximity based ads. Our smartphones are recently enabled to perform payments, just like a regular credit card.

All these new possibilities open up for a new kind of recommender systems, the demand being faster than academic response. The Google Context Awareness API<sup>2</sup> started to provide high quality, already processed location and contextual signals, enabling the applications to make use of them in a very simple way. A use case, for example, is to ask for a review of a restaurant the moment users move away from them. There are two different set of APIs that are provided: the Snapshot API gets instant details about the user current environment, while the Fence API reacts to changes in the user environment, giving the possibility to developers to create *contextual triggers*. The use of these APIs can provide *context-driven* algorithms the data they need for performing recommendations. If we can achieve to store this data in an anonymized way not in contrast with new EU regulations, we can give the freedom to the algorithms to exploit this huge amount of contextual information, as well as easiness of handling them, protecting the privacy of the users at the same time. Moreover a context-driven recommender can also be used as a fall-back algorithm in a personalized setting when users are not logged in the system. As an example, in the hotel booking domain, many users do not log in until the moment of payment. Another benefit of CDRS is relative to group recommendation, since usually a group of people share their context. In addition to that, context-driven algorithms can explain recommendation much more effectively: each recommendation can be explained by using the predominant *contextual signal* that led to that particular recommendation. Lastly, context-driven algorithms can actually “pop” the filter bubble [103], because the past history of the user does not prevent her to find new items of interest. Of course, contextualization has its own bubble effect, but people have more control over the situation and the goal of the moment than on their past history.

The context-driven paradigm requires to address a number of research challenges. Contextual variables, for instance, are often continuous, but factorization methods require the input to be discrete. So some methods have to be explored in order to transform the continuous input into a discrete form without losing information. An initial step into this direction has been made by [61]. Missing values also need to be addressed. Current state-of-the-art methods must have an input for each dimension, making it hard to handle missing contexts. While for some domains this does not constitute an issue, sensory data very often present missing values. The scalability of the recommendation phase poses some challenges to context-driven recommender systems. While for standard user-item setting recommendations can be precomputed, with context-driven is not so simple. Since the recommendation has to take into account each contextual state, a context-driven algorithm can either precompute them if the set of contextual states is not large or it has to compute recommendations in real time, given the set of contextual variables. Another issue is related to the evaluation: standard approaches can still be used but replacing the user with contextual variables in the simplest case. However,

---

<sup>2</sup><https://developers.google.com/awareness/>

when dealing with sessions, the evaluation must be adapted to take into account the sequential nature of the problem. There are some problems (like recommending the sequence of apps to perform a task) in which a strict ordering has to be respected. Other, instead, tolerate some degree of flexibility, like the playlist recommendation domain.

The conceptualization of the *contextual turn* [97] opens up for new challenges in recommender systems, allowing them to go beyond the classic, well explored, user-item matrix setting. We hope that such setting will foster some creative and out-of-the-box thinking about recommender systems. The potential for such a turn is already there: context-aware algorithms are already available, but they are tightly coupled with personalization; very powerful algorithms such as gradient boosted trees or deep learning are increasingly applied to traditional recommender systems problems, but they have just started to exploit the context.

In some domains, such as Jobs, Apps and Linear TV Program Recommendation, personalization could not be avoided and so the algorithms presented are not purely context-driven, but rather personalized algorithms heavily using context. So for some domains personalization is still a requirement, but there is room to exploit the context in a better way. Ensemble Theory has been proven to effectively take the best of many different algorithms (see Chapter 4 and Chapter 8). The power of the Ensemble Theory pictures a scenario where both personalization and contextualization can *coexist* and they both contribute to improve the serendipity, diversity and overall quality of recommendation. If the recommender systems community wants to keep up with industry needs, more realistic and different use case scenarios has to be addressed, instead of achieving very little improvement in a problem in which state-of-the-art techniques perform already well.

Concluding, research on recommender systems has always tried to mimic human behaviour in choosing items, but this was done by giving very little information to the algorithms. If we want the algorithms to mimic better the human *decision making process*, they have to take advantage of as many pieces of information as possible and know how to exploit them. This is the core idea of context-driven recommender systems: they try to infer as much as they can the situation and the intent of the user, exploiting all the available information that surrounds the event of a user interacting with a system. Only by providing the algorithm with nearly all the *decision variables* that the users implicitly uses to make a choice, we can expect algorithms to take decisions much more like humans do.

---

---

## Bibliography

---

- [1] Fabian Abel, András Benczúr, Daniel Kohlsdorf, Martha Larson, and Robert Pálovics. Recsys challenge 2016: Job recommendations. *Proceedings of ACM RecSys '16*.
- [2] Gediminas Adomavicius and Alexander Tuzhilin. *Context-Aware Recommender Systems*, pages 191–226. Springer, 2015.
- [3] Deepak Agarwal, Bee-Chung Chen, Rupesh Gupta, Joshua Hartman, Qi He, Anand Iyer, Sumanth Kolar, Yiming Ma, Pannagadatta Shivaswamy, Ajit Singh, et al. Activity ranking in linkedin feed. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1603–1612. ACM, 2014.
- [4] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.
- [5] Kamal Ali and Wijnand van Stam. TiVo: Making show recommendations using a distributed collaborative filtering architecture. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 394–401, New York, NY, USA, 2004. ACM.
- [6] Liliana Ardissono, Cristina Gena, Pietro Torasso, Fabio Bellifemine, Angelo Difino, and Barbara Negro. User modeling and recommendation techniques for personalized electronic program guides. In Liliana Ardissono, Alfred Kobsa, and T. Mark, editors, *Personalized Digital Television*, pages 3–26. Springer, Heidelberg, 2004.
- [7] Claudio Baccigalupo and Enric Plaza. Case-based sequential ordering of songs for playlist recommendation. In *Advances in Case-Based Reasoning*, pages 286–300. Springer, 2006.
- [8] Donghwan Bae, Keejun Han, Juneyoung Park, and Mun Y Yi. Apptrends: A graph-based mobile app recommendation system using usage history. In *Big Data and Smart Computing (BigComp), 2015 International Conference on*, pages 210–216. IEEE, 2015.

## Bibliography

---

- [9] Ricardo Baeza-Yates, Di Jiang, Fabrizio Silvestri, and Beverly Harrison. Predicting the next app that you are going to use. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 285–294. ACM, 2015.
- [10] Linas Baltrunas and Xavier Amatriain. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARS'09)*, 2009.
- [11] Linas Baltrunas, Karen Church, Alexandros Karatzoglou, and Nuria Oliver. Frappe: Understanding the usage and perception of mobile app recommendations in-the-wild. *arXiv preprint arXiv:1505.03014*, 2015.
- [12] Linas Baltrunas and Francesco Ricci. Context-based splitting of item ratings in collaborative filtering. In *Proceedings of the Third ACM Conference on Recommender Systems, RecSys '09*, pages 245–248, New York, NY, USA, 2009. ACM.
- [13] Linas Baltrunas and Francesco Ricci. Experimental evaluation of context-dependent collaborative filtering using item splitting. *User Modeling and User-Adapted Interaction*, 24(1-2):7–34, 2014.
- [14] Ana Belén Barragáns-Martínez, José J. Pazos-Arias, Ana Fernández-Vilas, Jorge García-Duque, and Martín López-Nores. What's on TV tonight? An efficient and effective personalized recommender system of TV programs. *IEEE Transactions on Consumer Electronics*, 55(1):286–294, 2009.
- [15] Mathieu Bastian, Matthew Hayes, William Vaughan, Sam Shah, Peter Skomoroch, Hyungjin Kim, Sal Uryasev, and Christopher Lloyd. LinkedIn skills: large-scale topic extraction and inference. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 1–8. ACM, 2014.
- [16] Alejandro Bellogin, Pablo Castells, and Ivan Cantador. Precision-oriented evaluation of recommender systems: An algorithmic comparison. In *RecSys '11*, pages 333–336, 2011.
- [17] David Ben-Shimon, Alexander Tsikinovsky, Michael Friedmann, Bracha Shapira, Lior Rokach, and Johannes Hoerle. Recsys challenge 2015 and the yoochoose dataset. In *Proceedings of the 9th ACM Conference on Recommender Systems, RecSys '15*, pages 357–358, New York, NY, USA, 2015. ACM.
- [18] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. *12th Int. Conf. on Music Information Retrieval (ISMIR)*, 2011.
- [19] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
- [20] Matthias Böhmer, Gernot Bauer, and Antonio Krüger. Exploring the design space of context-aware recommender systems that suggest mobile applications. In *2nd Workshop on Context-Aware Recommender Systems*, 2010.

- [21] Matthias Böhmer, Lyubomir Ganev, and Antonio Krüger. Appfunnel: A framework for usage-centric evaluation of recommender systems that suggest mobile applications. In *Proceedings of the 2013 international conference on Intelligent user interfaces*, pages 267–276. ACM, 2013.
- [22] Matthias Böhmer, Brent Hecht, Johannes Schöning, Antonio Krüger, and Gernot Bauer. Falling asleep with angry birds, facebook and kindle: a large scale study on mobile application usage. In *Proceedings of the 13th international conference on Human computer interaction with mobile devices and services*, pages 47–56. ACM, 2011.
- [23] Matthias Böhmer, Moritz Prinz, and Gernot Bauer. Contextualizing mobile applications for context-aware recommendation. *adj. proc. Of Pervasive*, 2010, 2010.
- [24] G Bonnin and D Jannach. Evaluating the quality of playlists based on hand-crafted samples. *14th Int. Society for Music Information Retrieval*, 2013.
- [25] Geoffray Bonnin and Dietmar Jannach. Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)*, 47(2):26, 2014.
- [26] Pia Borlund. The concept of relevance in IR. *JASIST*, 54(10):913–925, 2003.
- [27] Matthias Braunhofer, Mehdi Elahi, and Francesco Ricci. Sts: A context-aware mobile recommender system for places of interest. In *UMAP Workshops*. Cite-seer, 2014.
- [28] Robin D. Burke. Hybrid web recommender systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web, Methods and Strategies of Web Personalization*, pages 377–408. Springer, Heidelberg, 2007.
- [29] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.
- [30] Na Chang, Mhd Irvan, and Takao Terano. A TV program recommender framework. *Procedia Computer Science*, 22(0):561 – 570, 2013.
- [31] Zeina Chedrawy and Syed Sibte Raza Abidi. *A web recommender system for recommending, predicting and personalizing music playlists*. Springer, 2009.
- [32] Paul Cotter and Barry Smyth. PTV: Intelligent personalised TV guides. In *Proc. of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, pages 957–964. AAAI Press / The MIT Press, Cambridge, 2000.
- [33] Paolo Cremonesi, Franca Garzotto, Roberto Pagano, and Massimo Quadrana. Recommending without short head. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 245–246. International World Wide Web Conferences Steering Committee, 2014.

## Bibliography

---

- [34] Paolo Cremonesi, Franca Garzotto, and Massimo Quadrana. Evaluating top-n recommendations when the best are gone. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 339–342. ACM, 2013.
- [35] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010.
- [36] Paolo Cremonesi, Primo Modica, Roberto Pagano, Emanuele Rabosio, and Letizia Tanca. Personalized and context-aware tv program recommendations based on implicit feedback. In *International Conference on Electronic Commerce and Web Technologies*, pages 57–68. Springer, 2015.
- [37] Paolo Cremonesi, Roberto Pagano, Stefano Pasquali, and Roberto Turrin. TV program detection in tweets. In *Proceedings of the 11th european conference on Interactive TV and video*, pages 45–54. ACM, 2013.
- [38] Paolo Cremonesi and Roberto Turrin. Analysis of cold-start recommendations in IPTV systems. In *Proceedings of the third ACM conference on Recommender systems*, pages 233–236. ACM, 2009.
- [39] Yidong Cui and Kang Liang. A probabilistic top-n algorithm for mobile applications recommendation. In *Broadband Network & Multimedia Technology (IC-BNMT), 2013 5th IEEE International Conference on*, pages 129–133. IEEE, 2013.
- [40] Yidong Cui and Kang Liang. Top-n selection based on user behavior history formobile applications recommendation. *The Journal of China Universities of Posts and Telecommunications*, 20:1–6, 2013.
- [41] Sally Jo Cunningham, David Bainbridge, and Annette Falconer. More of an art than a science: Supporting the creation of playlists and mixes. In *Proceedings of 7th International Conference on Music Information Retrieval*, pages 240–245, Victoria, Canada, 2006.
- [42] Peter Danaher and Tracey Dagger. Using a nested logit model to forecast television ratings. *International Journal of Forecasting*, 28(3):607–622, 2012.
- [43] Peter J. Danaher, Tracey S. Dagger, and Michael S. Smith. Forecasting television ratings. *International Journal of Forecasting*, 27(4):1215 – 1240, 2011.
- [44] Duco Das and Herman ter Horst. Recommender systems for TV. 1998.
- [45] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.
- [46] Marcos Aurélio Domingues, Fabien Gouyon, Alípio Mário Jorge, José Paulo Leal, João Vinagre, Luís Lemos, and Mohamed Sordo. Combining usage and content in an online recommendation system for music in the long tail. *International Journal of Multimedia Information Retrieval*, 2(1):3–13, 2013.
- [47] Paul Dourish. What we talk about when we talk about context. *Personal and ubiquitous computing*, 8(1):19–30, 2004.



- [48] Susan T Dumais. Latent semantic analysis. *Annual review of information science and technology*, 38(1):188–230, 2004.
- [49] Alexandr Dzuba and Dmitry Bugaychenko. Mining users playbacks history for music recommendations. pages 422–430, 2014.
- [50] Benedikt Engelbert, Malte B. Blanken, Ralf Kruthoff-Brüwer, and Karsten Morisse. A user supporting personal video recorder by implementing a generic bayesian classifier based recommendation system. In *Proc. of PerCom Workshops*, pages 567–571. IEEE Computer Society, Los Alamitos, 2011.
- [51] Paul W Farris, Neil T Bendle, Phillip E Pfeifer, and David J Reibstein. *Marketing metrics: The definitive guide to measuring marketing performance*. Pearson Education, 2010.
- [52] Daniel Fleder and Kartik Hosanagar. Blockbuster culture’s next rise or fall: The impact of recommender systems on sales diversity. *Management science*, 55(5):697–712, 2009.
- [53] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, and Lars Schmidt-Thieme. Bayesian personalized ranking for non-uniformly sampled items. *JMLR W&CP, Jan*, 2012.
- [54] Charles Gouin-Vallerand and Neila Mezghani. An analysis of the transitions between mobile application usages based on markov chains. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 373–378. ACM, 2014.
- [55] Cathal Gurrin, Hyowon Lee, Paul Ferguson, Alan F. Smeaton, Noel E. O’Connor, Yoon-Hee Choi, and Heeseon Park. Social recommendation and visual analysis on the TV. In Alberto Del Bimbo, Shih-Fu Chang, and Arnold W. M. Smeulders, editors, *ACM Multimedia*, pages 1513–1514. ACM, 2010.
- [56] Negar Hariri, Bamshad Mobasher, and Robin Burke. Context-aware music recommendation based on latenttopic sequential patterns. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 131–138. ACM, 2012.
- [57] David Hauger, Markus Schedl, Andrej Kosir, and Marko Tkalcic. The million musical tweet dataset - what we can learn from microblogs. In *Proc. of the 14th Int. Society for Music Information Retrieval Conference*, Nov 4-8 2013.
- [58] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.
- [59] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 241–248. ACM, 2016.
- [60] Balázs Hidasi and Domonkos Tikk. Fast als-based tensor factorization for context-aware recommendation from implicit feedback. In *Proceedings of the*

## Bibliography

---

- 2012 *European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*, ECML PKDD'12, pages 67–82, Berlin, Heidelberg, 2012. Springer-Verlag.
- [61] Balázs Hidasi and Domonkos Tikk. Approximate modeling of continuous context in factorization algorithms. In *Proceedings of the 4th Workshop on Context-Awareness in Retrieval and Recommendation*, pages 3–9. ACM, 2014.
- [62] Shang H. Hsu, Ming-Hui Wen, Hsin-Chieh Lin, Chun-Chia Lee, and Chia-Hoang Lee. AIMED: A personalized TV recommendation system. In *Proceedings of the 5th European Conference on Interactive TV: A Shared Experience*, EuroITV'07, pages 166–174, Berlin, Heidelberg, 2007. Springer-Verlag.
- [63] Ke Huang, Chunhui Zhang, Xiaoxiao Ma, and Guanling Chen. Predicting mobile application usage using contextual information. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 1059–1065. ACM, 2012.
- [64] Rahul Jain, Joy Bose, and Tasleem Arif. Contextual adaptive user interface for android devices. In *India Conference (INDICON), 2013 Annual IEEE*, pages 1–5. IEEE, 2013.
- [65] Bo-Ram Jang, Yunseok Noh, Sang-Jo Lee, and Seong-Bae Park. A combination of temporal and general preferences for app recommendation. In *Big Data and Smart Computing (BigComp), 2015 International Conference on*, pages 178–185. IEEE, 2015.
- [66] Mingyu Joo, Kenneth C Wilbur, Bo Cowgill, and Yi Zhu. Television advertising and online search. *Management Science*, 60(1):56–73, 2013.
- [67] Alexandros Karatzoglou, Linas Baltrunas, Karen Church, and Matthias Böhmer. Climbing the app wall: enabling mobile app discovery through context-aware recommendations. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2527–2530. ACM, 2012.
- [68] Helen Katz. *The Media Handbook: A Complete Guide to Advertising Media Selection, Planning, Research, and Buying: A Complete Guide to Advertising Media Selection, Planning, Research, and Buying*. Routledge, 2014.
- [69] Christina M.L. Kelton and Linda G. Schneider Stone. Optimal television schedules in alternative competitive environments. *European Journal of Operational Research*, 104(3):451 – 473, 1998.
- [70] Jingu Kim and Taneli Mielikäinen. Conditional log-linear models for mobile application usage prediction. In *Machine Learning and Knowledge Discovery in Databases*, pages 672–687. Springer, 2014.
- [71] Yehuda Koren, Robert Bell, Chris Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.
- [72] Kaushal Kurapati, Srinivas Gutta, David Schaffer, Jacquelyn Martino, and John Zimmerman. A multi-agent tv recommender. In *Proc. of the 1st International Workshop on Personalization in Future TV*, 2001.

- 
- [73] Monle Lee and Carla Johnson. *Principles of advertising: a global perspective*. Routledge, 2013.
- [74] Lukas Lerche and Dietmar Jannach. Using graded implicit feedback for bayesian personalized ranking. In *ACM RecSys '14*, pages 353–356, 2014.
- [75] Zhung-Xun Liao, Shou-Chung Li, Wen-Chih Peng, Philip S Yu, and Te-Chuan Liu. On the feature discovery for app usage prediction in smartphones. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 1127–1132. IEEE, 2013.
- [76] Zhung-Xun Liao, Yi-Chin Pan, Wen-Chih Peng, and Po-Ruey Lei. On mining mobile apps usage behavior for predicting apps usage in smartphones. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 609–618. ACM, 2013.
- [77] Ning-Han Liu and Shu-Ju Hsieh. Intelligent music playlist recommendation based on user daily behavior and music content. pages 671–683, 2009.
- [78] Xin Liu. Towards context-aware social recommendation via trust networks. In *International Conference on Web Information Systems Engineering*, pages 121–134. Springer, 2013.
- [79] Lennart Ljung. *System identification: theory for the user*. Prentice-Hall., 1999.
- [80] Babak Loni, Roberto Pagano, Martha Larson, and Alan Hanjalic. Bayesian personalized ranking with multi-channel user feedback. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 361–364. ACM, 2016.
- [81] Babak Loni and Alan Said. Wraprec: an easy extension of recommender system libraries. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 377–378. ACM, 2014.
- [82] Babak Loni, Yue Shi, Martha Larson, and Alan Hanjalic. Cross-domain collaborative filtering with factorization machines. In *Proceedings of the 36th European Conference on Information Retrieval, ECIR '14*, 2014.
- [83] Eric Hsueh-Chan Lu, Yi-Wei Lin, and Jing-Bin Ciou. Mining mobile application sequential patterns for usage prediction. In *Granular Computing (GrC), 2014 IEEE International Conference on*, pages 185–190. IEEE, 2014.
- [84] Augusto Q Macedo, Leandro B Marinho, and Rodrygo LT Santos. Context-aware event recommendation in event-based social networks. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 123–130. ACM, 2015.
- [85] Brian McFee, Luke Barrington, and Gert Lanckriet. Learning content similarity for music recommendation. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(8):2207–2218, 2012.
- [86] Lei Meng, Shu Liu, and Aaron Striegel. Analyzing the longitudinal impact of proximity, location, and personality on smartphone usage. *Computational Social Networks*, 1(1):1–15, 2014.

## Bibliography

---

- [87] K. Metaxiotis, K. Nikolopoulos, V. Assimakopoulos, and A. Patelis. FORTV: decision support system for forecasting television viewership. *Journal of Computer Information Systems*, 43(4):100–107, 2003.
- [88] Denny Meyer and Rob J Hyndman. The accuracy of television network rating forecasts: The effects of data aggregation and alternative models. *Model Assisted Statistics and Applications*, 1(3):147–155, 2006.
- [89] Stefano Mizzaro, Marco Pavan, Ivan Scagnetto, and Ivano Zanello. A context-aware retrieval system for mobile applications. In *Proceedings of the 4th Workshop on Context-Awareness in Retrieval and Recommendation*, pages 18–25. ACM, 2014.
- [90] Alistair Moffat. Implementing the ppm data compression scheme. *IEEE Transactions on communications*, 38(11):1917–1921, 1990.
- [91] Douglas C Montgomery, Cheryl L Jennings, and Murat Kulahci. *Introduction to time series analysis and forecasting*, volume 526. John Wiley & Sons, 2011.
- [92] Simon Murray. TV advertising forecasts. <http://www.digitaltvresearch.com/products/product?id=107>, August 2014. (accessed April 2015).
- [93] Trung V. Nguyen, Alexandros Karatzoglou, and Linas Baltrunas. Gaussian process factorization machines for context-aware recommendations. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR '14*, pages 63–72, New York, NY, USA, 2014. ACM.
- [94] Nielsen. Nielsen’s 2015 advance national tv household universe estimate (UE), April 2015.
- [95] Konstantinos Nikolopoulos, P. Goodwin, A. Patelis, and V. Assimakopoulos. Forecasting with cue information: A comparison of multiple regression with alternative forecasting approaches. *European Journal of Operational Research*, 180(1):354–368, 2007.
- [96] Jinoh Oh, Sungchul Kim, Jinha Kim, and Hwanjo Yu. When to recommend: A new issue on TV show recommendation. *Information Sciences*, 280(0):261 – 274, 2014.
- [97] Roberto Pagano, Paolo Cremonesi, Martha Larson, Balázs Hidasi, Domonkos Tikk, Alexandros Karatzoglou, and Massimo Quadrana. The contextual turn: From context-aware to context-driven recommender systems. In *Proceedings of the 10th ACM Conference on Recommender Systems, RecSys '16*, pages 249–252, New York, NY, USA, 2016. ACM.
- [98] Roberto Pagano, Massimo Quadrana, Paolo Cremonesi, Sergio Bittanti, Simone Formentin, and Andrea Mosconi. Prediction of TV ratings with dynamic models. *ACM Workshop on Recommendation Systems for Television and Online Video (RecSysTV 2015)*.

- 
- [99] Weike Pan, Hao Zhong, Congfu Xu, and Zhong Ming. Adaptive bayesian personalized ranking for heterogeneous implicit feedbacks. *Knowledge-Based Systems*, 73:173 – 180, 2015.
- [100] Umberto Panniello, Alexander Tuzhilin, and Michele Gorgoglione. Comparing context-aware recommender systems in terms of accuracy and diversity. *User Modeling and User-Adapted Interaction*, 24(1-2):35–65, 2014.
- [101] Umberto Panniello, Alexander Tuzhilin, Michele Gorgoglione, Cosimo Palmisano, and Anto Pedone. Experimental comparison of pre- vs. post-filtering approaches in context-aware recommender systems. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 265–268, New York, NY, USA, 2009. ACM.
- [102] Abhinav Parate, Matthias Böhmer, David Chu, Deepak Ganesan, and Benjamin M Marlin. Practical prediction and prefetch for faster access to applications on mobile phones. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 275–284. ACM, 2013.
- [103] Eli Pariser. *The filter bubble: What the Internet is hiding from you*. Penguin UK, 2011.
- [104] Robi Polikar. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3):21–45, 2006.
- [105] Runwei Qiang, Feng Liang, and Jianwu Yang. Exploiting ranking factorization machines for microblog retrieval. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, CIKM '13, pages 1783–1788, New York, NY, USA, 2013. ACM.
- [106] Robert Ragno, Christopher JC Burges, and Cormac Herley. Inferring similarity between music objects with application to playlist generation. In *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 73–80. ACM, 2005.
- [107] Srinivas K. Reddy, Jay E. Aronson, and Antonie Stam. Spot: Scheduling programs optimally for television. *Management Science*, 44(1):83–102, 1998.
- [108] Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pages 995–1000. IEEE, 2010.
- [109] Steffen Rendle. Factorization machines with libfm. *ACM Trans. Intell. Syst. Technol.*, 3(3), May 2012.
- [110] Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 273–282. ACM, 2014.
- [111] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

## Bibliography

---

- [112] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. *SIGIR '11*. ACM, 2011.
- [113] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.
- [114] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.
- [115] Alan Said, Babak Loni, Roberto Turrin, and Andreas Lommatzsch. An extended data model format for composite recommendation. In *Proc. of the 8th RecSys conference 2014*, Foster City, Silicon Valley, CA, USA, 2014.
- [116] Markus Schedl and Dominik Schnitzer. Location-aware music artist recommendation. *MultiMedia Modeling*, pages 1–9, 2014.
- [117] Kent Shi and Kamal Ali. Getjar mobile application recommendations with very sparse datasets. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 204–212. ACM, 2012.
- [118] Yue Shi, Martha Larson, and Alan Hanjalic. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys*, 2014.
- [119] Choonsung Shin, Jin-Hyuk Hong, and Anind K Dey. Understanding and prediction of mobile application usage for smart phones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 173–182. ACM, 2012.
- [120] Joseph Sill, Gábor Takács, Lester Mackey, and David Lin. Feature-weighted linear stacking. *arXiv preprint arXiv:0911.0460*, 2009.
- [121] Barry Smyth and Paul Cotter. Surfing the digital wave, generating personalised TV listings using collaborative, case-based recommendation. In *Althoff K.D., Bergmann R., Branting K. (eds), 'Case-Based Reasoning Research and Development, Proceedings of the Third International Conference on Case-Based Reasoning ICCBR99*, pages 561–571. Springer Verlag, 1999.
- [122] Vijay Srinivasan, Saeed Moghaddam, Abhishek Mukherji, Kiran K Rachuri, Chenren Xu, and Emmanuel Munguia Tapia. Mobileminer: Mining your frequent patterns on your phone. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 389–400. ACM, 2014.
- [123] Li-Yu Tang, Pi-Cheng Hsiu, Jiun-Long Huang, and Ming-Syan Chen. ilauncher: an intelligent launcher for mobile apps based on individual usage patterns. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 505–512. ACM, 2013.
- [124] James T Tiedge and Kenneth J Ksobiech. The “lead-in” strategy for prime-time tv: Does it increase the audience? *Journal of Communication*, 36(3):51–63, 1986.

- 
- [125] Roberto Turrin, Andrea Condorelli, Paolo Cremonesi, and Roberto Pagano. Time-based TV programs prediction. In *1st Workshop on Recommender Systems for Television and Online Video at ACM RecSys*, 2014.
- [126] Roberto Turrin, Andrea Condorelli, Paolo Cremonesi, Roberto Pagano, and Massimo Quadrona. Large scale music recommendation. In *ACM Workshop on Large-Scale Recommender Systems (LSRS 2015)*.
- [127] Roberto Turrin, Massimo Quadrona, Andrea Condorelli, Roberto Pagano, and Paolo Cremonesi. 30music listening and playlists dataset. In *Poster Proceedings of the 9th ACM Conference on Recommender Systems (RecSys 2015)*.
- [128] Christian Überall, Rajarajan Muttukrishnan, Veselin Rakocevic, Rudolf Jäger, and Christopher Köhnen. Recommendation index for DVB content using service information. In *Proceedings of the 2009 IEEE International Conference on Multimedia and Expo, ICME'09*, pages 1178–1181, Piscataway, NJ, USA, 2009. IEEE Press.
- [129] Shoko Wakamiya, Ryong Lee, and Kazutoshi Sumiya. Towards better TV viewing rates: exploiting crowd's media life logs over twitter for tv rating. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*, page 39. ACM, 2011.
- [130] Qiudang Wang, Xiao Liu, Shasha Zhang, Yuanchun Jiang, Fei Du, Yading Yue, and Yu Liang. A novel app recommendation method based on svd and social influence. In *Algorithms and Architectures for Parallel Processing*, pages 269–281. Springer, 2015.
- [131] Sheng Wang, Xiaobo Zhou, Ziqi Wang, and Ming Zhang. Please spread: Recommending tweets for retweeting with implicit feedback. In *Proceedings of the 2012 Workshop on Data-driven User Behavioral Modelling and Mining from Social Media, DUBMMSM '12*, pages 19–22, New York, NY, USA, 2012. ACM.
- [132] William Wu-Shyong Wei. *Time series analysis*. Addison-Wesley Redwood City, California, 1994.
- [133] Shi Wenxuan and Yin Airu. Interoperability-enriched app recommendation. In *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*, pages 1242–1245. IEEE, 2014.
- [134] Kenneth C Wilbur. A two-sided, empirical model of television advertising and viewing markets. *Marketing Science*, 27(3):356–378, 2008.
- [135] Xiao Xia, Xiaodong Wang, Jian Li, and Xingming Zhou. Multi-objective mobile app recommendation: A system-level collaboration approach. *Computers & Electrical Engineering*, 40(1):203–215, 2014.
- [136] Xiao Xia, Xiaodong Wang, Xingming Zhou, and Bo Liu. Evolving mobile app recommender systems: An incremental multi-objective approach. In *Future Information Technology*, pages 21–27. Springer, 2014.

## Bibliography

---

- [137] Xiao Xia, Xiaodong Wang, Xingming Zhou, and Tao Zhu. Collaborative recommendation of mobile apps: A swarm intelligence method. In *Mobile, Ubiquitous, and Intelligent Computing*, pages 405–412. Springer, 2014.
- [138] Ye Xu, Mu Lin, Hong Lu, Giuseppe Cardone, Nicholas Lane, Zhenyu Chen, Andrew Campbell, and Tanzeem Choudhury. Preference, context and communities: a multi-faceted approach to predicting smartphone app usage patterns. In *Proceedings of the 2013 International Symposium on Wearable Computers*, pages 69–76. ACM, 2013.
- [139] Bo Yan and Guanling Chen. Appjoy: personalized mobile application discovery. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 113–126. ACM, 2011.
- [140] Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 113–126. ACM, 2012.
- [141] Cheng Yang, Tao Wang, Gang Yin, Huaimin Wang, Ming Wu, and Ming Xiao. Personalized mobile application discovery. In *Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, pages 49–54. ACM, 2014.
- [142] Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. App recommendation: a contest between satisfaction and temptation. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 395–404. ACM, 2013.
- [143] Peng Yu and Ching-man Au Yeung. App mining: finding the real value of mobile applications. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 417–418. International World Wide Web Conferences Steering Committee, 2014.
- [144] Yong Zheng, Bamshad Mobasher, and Robin D Burke. The role of emotions in context-aware recommendation. *Decisions@ RecSys*, 2013:21–28, 2013.
- [145] Konglin Zhu, Xiaoyi Zhang, Bin Xiang, and Lin Zhang. Exploiting user context and network information for mobile application usage prediction. In *Proceedings of the 7th International Workshop on Hot Topics in Planet-scale mObile computing and online Social neTworking*, pages 25–30. ACM, 2015.
- [146] Taojiang Zhu, Jiabao Yan, and Ying Zhao. What special about top-n recommendation for mobile app stores. In *Information Technology and Electronic Commerce (ICITEC), 2014 2nd International Conference on*, pages 306–310. IEEE, 2014.
- [147] John Zimmerman, Kaushal Kauapati, AnnaL. Buczak, Dave Schaffer, Srinivas Gutta, and Jacquelyn Martino. TV personalization system. In *Personalized Digital Television*, volume 6 of *Human-Computer Interaction Series*, pages 27–51. Springer Netherlands, 2004.



- [148] Xun Zou, Wangsheng Zhang, Shijian Li, and Gang Pan. Prophet: What app you wish to use next. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, pages 167–170. ACM, 2013.