



**FACULTY  
OF INFORMATION  
TECHNOLOGY  
CTU IN PRAGUE**

## ASSIGNMENT OF BACHELOR'S THESIS

**Title:** Analyzing Impact of Interaction Context in Collaborative Filtering  
**Student:** Martin Scheubrein  
**Supervisor:** Ing. Tomáš Řehořek  
**Study Programme:** Informatics  
**Study Branch:** Knowledge Engineering  
**Department:** Department of Applied Mathematics  
**Validity:** Until the end of summer semester 2019/20

### Instructions

Survey the area of Recommender Systems, mainly the methods of Collaborative Filtering and their offline evaluation (e.g., Recall, Catalog Coverage). Propose different ways of affecting the models by various contexts in which they are being trained and evaluated, such as the user's device, the season of the year, time elapsed since last interactions, etc. Design a framework for the evaluation of different CF algorithms and contexts in a parametrizable way. Implement the framework in the language of your choice, including selected algorithms and proposed improvements. On provided datasets conduct set of experiments with implemented algorithms and their different contextual parametrizations. Evaluate the impact of various contexts on performance and discuss the results.

### References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.  
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.  
Dean

Prague January 23, 2019





**CZECH TECHNICAL  
UNIVERSITY  
IN PRAGUE**

**FIT**

**Faculty of Information Technology  
Department of Applied Mathematics**

**Bachelor's Thesis**

# **Analyzing Impact of Interaction Context in Collaborative Filtering**

**Martin Scheubrein**

**May 2019**

**Supervisor: Ing. Tomáš Řehořek, Ph.D.**





## Acknowledgement / Declaration

Rád bych poděkoval zejména svému vedoucímu, Ing. Tomáši Řehořkovi, Ph.D., za uvedení do oblasti doporučování, za vedení a motivaci. Dále děkuji společnosti Recombee za zapůjčení výpočetního uzlu pro experimenty. Děkuji i všem dalším, zpravidla duševním podporovatelům.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 13. 5. 2019

.....



## Abstrakt / Abstract

Kolaborativní filtrování je jednou z nejúspěšnějších technik používaných v doporučovacích systémech. Základní algoritmy využívají historické interakce mezi uživateli a předměty, nicméně doporučovací systémy nasazené v produkčním prostředí mají často k dispozici minimálně jednu další dimenzi dat – časová razítka těchto interakcí. Tyto okolnosti interakcí nazýváme *kontextem*.

Tato práce využívá dosud často opomíjené informace v datech ke zlepšení přesnosti doporučování. Je navrženo několik nových přístupů k začlenění kontextu do tradičních metod kolaborativního filtrování. K evaluaci těchto vylepšení je navržen a implementován testovací framework. Navržené metody jsou rozsáhle testovány na několika datasetech, s různými parametry a kontexty.

Výsledky ukazují, že metody beroucí v úvahu kontext vykazují i na převážně statických datasetech zlepšení metriky *recall* o 5–25 % oproti tradičním algoritmům kolaborativního filtrování.

**Klíčová slova:** doporučovací systémy, kolaborativní filtrování, kontext, časová dynamika

Collaborative filtering is one of the most successful techniques used in recommender systems. The basic algorithms utilize history of interactions between users and items. However, recommenders deployed in production often have at least one more dimension of data available—timestamp of the interaction. These interaction circumstances are collectively referred to as *the context*.

This thesis exploits the additional information in order to improve overall recommender accuracy. Several novel approaches to incorporating context into traditional collaborative filtering are proposed. An evaluation framework is designed and proposed algorithms are extensively evaluated with different parameters and contexts on multiple datasets.

Results show that even mostly static datasets benefit from the proposed context-aware approach. About 5–35 % recall increase was observed in comparison with traditional collaborative filtering algorithms.

**Keywords:** recommender systems, collaborative filtering, interaction context, temporal dynamics





# / Contents

<b>1 Introduction</b> .....	1
<b>2 Recommender Systems</b> .....	3
2.1 Content-Based Recommending ..	3
2.2 Collaborative Filtering.....	4
2.2.1 User $k$ -NN.....	5
2.2.2 Item $k$ -NN .....	6
<b>3 Design of a Context-Aware Recommender</b> .....	7
3.1 Related Research.....	7
3.2 Proposed Algorithms .....	8
3.2.1 Temporal Dynamics .....	8
3.2.2 Discrete Segmentation ...	10
3.3 Evaluation Framework.....	12
<b>4 Experiments</b> .....	14
4.1 Performance Metrics.....	14
4.2 Testing Datasets .....	15
4.3 Fixed Parameters .....	16
4.4 Item Significance.....	17
4.5 Temporal Dynamics .....	17
4.5.1 Weighting the User Vectors by Global End- of-Time.....	18
4.5.2 Weighting the Rating Matrix by Global End- of-Time.....	19
4.5.3 Weighting the User Vectors by User- Specific End-of-Time ....	20
4.5.4 Weighting the Rat- ing Matrix by User- Specific End-of-Time ....	22
4.6 Discrete Segmentation .....	24
4.6.1 Furniture Dataset.....	26
4.6.2 Posters Dataset .....	27
4.6.3 Outdoor Dataset .....	28
4.6.4 Streaming Dataset .....	30
4.6.5 Bagging .....	32
<b>5 Conclusion</b> .....	34
<b>References</b> .....	36
<b>A The Evaluation Framework and Measurements</b> .....	39
A.1 The Evaluation Framework....	39
A.2 Measurements.....	40



## Tables / Figures

<p><b>4.1.</b> Rating matrix dimensions of the experimental datasets..... 16</p> <p><b>4.2.</b> Reference coverage/recall for User 25-NN and Item 10-NN .. 25</p> <p><b>4.3.</b> Recall of context-aware User 25-NN and Item 10-NN algorithms on Furniture dataset ... 27</p> <p><b>4.4.</b> Recall of context-aware User 25-NN and Item 10-NN algorithms on Posters dataset . 28</p> <p><b>4.5.</b> Recall of context-aware User 25-NN and Item 10-NN algorithms on Outdoor dataset .... 29</p> <p><b>4.6.</b> Recall of context-aware User 25-NN and Item 10-NN algorithms on Streaming dataset .. 32</p> <p><b>4.7.</b> Comparison of selected recall measurements with recall of all-ones matrices ..... 33</p>	<p><b>3.1.</b> Temporal weighting functions . 10</p> <p><b>3.2.</b> Identity matrix examples ..... 11</p> <p><b>3.3.</b> Schema of the evaluation framework ..... 13</p> <p><b>4.1.</b> Impact of model size to recall/coverage ..... 16</p> <p><b>4.2.</b> Impact of testing/training set size ratio on measured recall and coverage ..... 17</p> <p><b>4.3.</b> Distribution of item popularity in a dataset ..... 18</p> <p><b>4.4.</b> User <math>k</math>-NN weighting applied to user vectors with <math>t_0</math> ..... 19</p> <p><b>4.5.</b> Item <math>k</math>-NN weighting applied to user vectors with <math>t_0</math> ..... 20</p> <p><b>4.6.</b> Exponential weighting applied to rating matrix with <math>t_0</math> ..... 21</p> <p><b>4.7.</b> Different User <math>k</math>-NN weighting functions applied to user vectors with <math>t_0(u)</math> ..... 22</p> <p><b>4.8.</b> Different Item <math>k</math>-NN weighting functions applied to user vectors with <math>t_0(u)</math> ..... 23</p> <p><b>4.9.</b> User <math>k</math>-NN with exponential weighting applied to the rating matrix with <math>t_0(u)</math> ..... 24</p> <p><b>4.10.</b> Various Item <math>k</math>-NN weighting applied to the rating matrix with <math>t_0(u)</math>..... 25</p> <p><b>4.11.</b> Similarity matrices of natural periodicities in Furniture dataset..... 26</p> <p><b>4.12.</b> Similarity matrices of natural periodicities in Posters dataset..... 28</p> <p><b>4.13.</b> Similarity matrix of natural periodicity in Outdoor dataset..... 29</p> <p><b>4.14.</b> Similarity matrices of the Streaming dataset for (a) season, (b) month of a year.... 30</p> <p><b>4.15.</b> Day of week similarity matrices of the Streaming dataset... 31</p> <p><b>4.16.</b> Hour similarity matrices of the Streaming dataset ..... 31</p>
---	---

# Chapter 1

## Introduction

Recommender systems play an important role in navigation on modern Internet. They complement the palette of information retrieval tools, guiding users to content which is interesting for them. Recommender systems help users to discover relevant items even if they exactly do not know what they are looking for. They are of significant business value especially for E-commerce and for content providers.

Collaborative filtering is a class of recommender algorithms utilizing the behaviour of other users known to the system to predict some items to the target user. This thesis focuses on two most basic, yet widely and successfully used collaborative filtering algorithms, User  $k$ -NN and Item  $k$ -NN.

The standard versions of recommender algorithms don't take advantage of all information available in production datasets. Often, the timestamp of an interaction of a user with an item is known, because it can be obtained trivially and without the explicit action from the user. For this reason, several options of exploiting the knowledge of timestamps and incorporating it into collaborative filtering algorithms are suggested. A framework capable of measuring the performance of the proposed improvements with various parameters is designed and implemented. The proposed methods are thoroughly tested under miscellaneous circumstances and combinations of parameters using the framework.

This thesis aims to confirm the assumption that the time context plays an important role and influences the recommendation task heavily. User preferences are known to be dynamic and previous research has shown that time information can be used to increase the recommendation accuracy. The thesis extends the existing attempts to create a context-aware recommender and experiments with multiple values of model parameters in order to find the combination leading to the greatest improvement of chosen accuracy metrics.

As a side effect, a method of visualization of periodicity in collaborative filtering datasets is introduced.

The contributions of this thesis over previous related research include:

- Focus on implicit datasets, for which it is generally harder to make recommendations. Previous research often studies datasets with explicit users' ratings of the items, but business data usually contains only implicit ratings. Therefore, our research is more useful for business operators of recommender systems.
- Wide spectrum of parametrization options and a large number of experiments with different parameter combinations.
- Comparison of behaviour of the same algorithms on multiple datasets with different characteristics, preventing from overfitting to a specific dataset.
- An innovative method of dataset periodicity visualization.
- All proposed methods are basically wrappers for the standard algorithms, allowing extension of possibly existing implementations of collaborative filtering algorithms easily and the modifications are not tied to a single base algorithm.

In Chapter 2, general theoretical background of recommender systems is presented. Emphasis is put on collaborative filtering methods and two particular algorithms, User  $k$ -NN and Item  $k$ -NN are described in detail. Chapter 3 presents the problem of temporal dynamics of user preferences and their dependence on context. Several modifications of standard collaborative filtering algorithms are proposed in order to exploit the contextual information. An evaluation framework capable of measuring the influence on recommendation accuracy is designed. In Chapter 4, an extensive set of experiments is performed and the accuracy of the basic algorithms is compared to the improved algorithms. Chapter 5 concludes the results of the experiments and assesses the suitability of the proposed methods.

## Chapter 2

# Recommender Systems

A recommender system works with two kinds of entities—*users* and *items*. A *user* is a uniquely identified person or a bot accessing a website. It may be a person registered under an online account, or it may be a single browser session, whatever the website is capable of collecting. The term *item* denotes a class of product the user can interact with. Both users and items may have attributes of any kind (textual description, image, numeric attributes, etc.).

A recommender system is a predictive model which tries to score each available item for a given user. The higher the score is, the higher should be the probability of usefulness of the item to the user. This task is often simplified to a problem of finding only top- $n$  items with the highest scores and often the precise value of a score is not even needed. This is the case for online retailers, where such system *recommends* a few products the user may be interested in, with  $n$  being usually around 5–10. In later sections, only the first variant (score each item) is discussed, as the top- $n$  recommendation could be directly derived from a list of scored items.

Two most widely used types of recommender systems are *content-based recommendation* and *collaborative filtering* (CF). In content-based recommendation, the prediction is based on attribute similarities between items. In collaborative filtering, the prediction is based on historic interactions between users and items and doesn't need neither user nor item attributes at all, except for a unique id. Due to practical reasons, which will be covered in later sections, it may be beneficial to combine these two approaches and form a *hybrid recommender system*.

### 2.1 Content-Based Recommending

Content-based recommendation builds on a hypothesis that if a user likes an item, he will like similar items, too. This method relies on item attributes heavily. It may involve user attributes as well.

Since attributes can be of many datatypes, there are countless similarity measures defined between two items. The most common attribute is a textual description, which opens possibility to use well-known algorithms of information retrieval, such as *bag of words* or *tf-idf* [1, p. 79–81]. There is a wide range of preprocessing options, e.g. *stop-word* removal, *stemming*, and more. Other attribute types include structured data, which are easier to process. Numeric or ordinal attributes (price, power consumption, distance to nearest shop) can be compared in terms of metric distance/similarity, whereas other nominal attributes (category, film genre) can be compared for example using Jaccard similarity index. Image attributes can be compared using their colour histograms, SIFT features [2] or using an autoencoder artificial neural network to encode image data into vectors and then computing a cosine similarity of those encodings.

Having constructed an appropriate similarity measure, the algorithm works as follows. In training phase, the algorithm remembers items and their attributes, possibly

building a metric index for faster future retrieval. In the inference phase, the algorithm scores each possible item by choosing  $k$  nearest neighbours (with respect to chosen similarity function) from items with which the user has interacted in the past, and it derives its score from known scores of these  $k$  neighbours [3] e.g. as a weighted average of these scores.

Content-based recommenders tend to suffer from overspecialization [4, p. 1184], which means they recommend items too similar to be useful. In practice, it is desirable to provide more *long-tail* recommendations, enabling the user to discover new content [5, p. 2].

## 2.2 Collaborative Filtering

To describe the algorithms precisely, it is needed to define some concepts formally. Let  $U$  be the ordered set of all users and  $I$  be the ordered set of all items. Let  $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{|U| \times |I|}$  be a matrix with  $|U|$  rows and  $|I|$  columns, where each element is either a real number or an artificial symbol “?”. Such matrix is called *the rating matrix* or *the user-item matrix*. Element  $\mathbf{R}_{u,i}$  contains a number expressing opinion of user  $u \in U$  on item  $i \in I$ , where higher number means more positive relationship between user and item. Symbol “?” denotes unknown rating.

Rating matrix  $\mathbf{R}$  is very sparse, which means it contains mostly “?” symbols. In a recommender system, there are usually thousands to millions of items, while a typical user has interacted with only few, rarely with more than a hundred.

Nature of known ratings may vary from dataset to dataset. Some webpages allow users to rate items explicitly (e.g. users of a film database may rate films by giving them one to five stars) and this type of feedback provides the most accurate overview of user’s taste. It may express both liking and disliking. Downside of explicit ratings is that they require additional effort from a user and users are usually not willing to take action to give explicit feedback [6, p. 263], thus they are not always available. Implicit ratings are much easier to gather because no change in user’s behaviour is needed. Implicit ratings include click-through, cart addition, purchase, time spent looking at a description, video play etc. This data is used to estimate user’s attitude to an item. Implicit ratings are a natural way to get feedback from users, but unfortunately, they are mostly binary (a user either plays or doesn’t play a video), hence they fill the rating matrix only with positive or unknown ratings, not taking negative ones into consideration.

Implicit ratings, however, often carry a lot of additional information, such as timestamp of an interaction, device from which the interaction has been performed, or screen resolution. These circumstances, termed *context* of the interaction, are not used in traditional recommender systems. This thesis examines possibilities of incorporation of context into collaborative filtering algorithms.

In the following sections, two prominent CF algorithms are described in detail. As they need to do arithmetics on the rating matrix, they both assume all “?” values are converted to zero before computation, which is a common way to deal with unknown values, since zero expresses neutral attitude of a user to an item. There are other popular algorithms in use, namely the family of matrix factorization methods [7, p. 113], but they are not covered by this thesis.



### 2.2.1 User $k$ -NN

User  $k$ -NN ( $k$  nearest neighbours), also called user-based collaborative filtering, together with Item  $k$ -NN, are the most basic yet powerful and widely used algorithms of collaborative filtering. As the term *collaborative* suggests, interactions of all users are used when predicting for a given user. User  $k$ -NN works with a hypothesis that the user will like the same items as users who had similar preferences in the past.

When scoring an item  $i \in I$  for target user  $u \in U$ , the algorithm selects a set of  $k \in \mathbb{N}$  most similar users to user  $u$ , denoted  $\text{NN}_k(u)$ , according to given similarity measure  $\text{sim} : I^2 \rightarrow \mathbb{R}$ . Particular choice of the similarity function and parameter  $k$  will be discussed later. The predicted score for user  $u$  and item  $i$  is then computed as a weighted average [8, p. 13–16]:

$$\text{pred}(u, i) = \overline{\mathbf{R}_{u,*}} + \frac{\sum_{v \in \text{NN}_k(u)} \text{sim}(u, v) \cdot (\mathbf{R}_{v,i} - \overline{\mathbf{R}_{v,*}})}{\sum_{v \in \text{NN}_k(u)} \text{sim}(u, v)}$$

Symbol  $\overline{\mathbf{R}_{u,*}}$  denotes average rating of user  $u$ . If rating matrix  $\mathbf{R}$  contains only implicit ratings (which is the case for datasets used in experiments in Chapter 4), they can't be biased by user's personal perception of rating scale, so it is not needed to reflect the average ratings. The formula then simplifies to

$$\text{pred}(u, i) = \frac{\sum_{v \in \text{NN}_k(u)} \text{sim}(u, v) \cdot \mathbf{R}_{v,i}}{\sum_{v \in \text{NN}_k(u)} \text{sim}(u, v)}$$

Parameter  $k$  is called *model size* and affects both computational performance and accuracy of the algorithm. Low values of  $k$  lead to overfitting and recommendation quality is poor. For larger values of  $k$ , accuracy improves, however memory requirements grow. Very large values of  $k$  lead to underfitting and accuracy decrease. Results of [9] show that reasonable values of  $k$  are 20–50 and experiments conducted for purposes of this thesis confirm these numbers.

Some common choices of similarity measure are Pearson's correlation coefficient, cosine similarity or Spearman's rank correlation coefficient. For this thesis, cosine similarity has been chosen due to its simplicity and widespreadness. More precisely, implementation of User  $k$ -NN used in experiments in Chapter 4 uses similarity function

$$\text{sim}(u, v) = \frac{\sum_{i \in I} \mathbf{R}_{u,i} \cdot \mathbf{R}_{v,i}}{\sqrt{\sum_{i \in I} \mathbf{R}_{u,i}^2} \cdot \sqrt{\sum_{i \in I} \mathbf{R}_{v,i}^2}}$$

Drawbacks of User  $k$ -NN are mainly the *cold-start* problem and scalability. Cold-start problem appears when there are only a few interactions in a recommender system. In such system, similarities between users are often zero, therefore the algorithm is not able to find a relevant set of  $k$  most similar users, which results in poor quality recommendation. This also happens on smaller scale when a new user or item appears in a system. In order to deal with cold-start problem, techniques used in content-based recommendation may be used when there is not enough information for User  $k$ -NN. Scalability is also an issue, because  $|U|$  and  $|I|$  can easily exceed millions and searching for  $k$  most similar users becomes computationally very expensive.

### 2.2.2 Item $k$ -NN

Item  $k$ -NN is another important algorithm of collaborative filtering. It is related to User  $k$ -NN, and we can roughly think of these two as transpositions of each other. The fundamental assumption of Item  $k$ -NN is that if a user likes item  $i_1$  and a lot of users who like  $i_1$  also like another item  $i_2$ , then the target user will like  $i_2$ , too.

The algorithm works as follows. For each item  $j$  for which there is a known rating  $\mathbf{R}_{u,j} \in \mathbb{R}$  from the user  $u$ , find a set of  $k$  most similar items  $\text{NN}_k(j)$ . The score for item  $i \in I$  is computed as an average of user's known ratings weighted by similarities of items  $i$  and  $j$  [10]. The items which are not present in any of the  $\text{NN}_k(j)$  sets will be scored zero. If  $\text{KR}(u)$  denotes known ratings of user  $u$ , the prediction for item  $i$  is:

$$\text{pred}(u, i) = \begin{cases} \frac{\sum_{\substack{i \in \text{NN}_k(j) \\ j \in \text{KR}(u)}} \text{sim}(i, j) \cdot \mathbf{R}_{u, j}}{\sum_{\substack{i \in \text{NN}_k(j) \\ j \in \text{KR}(u)}} \text{sim}(i, j)} & \text{if } \exists j \in \text{KR}(u) : i \in \text{NN}_k(j) \\ 0 & \text{if } \forall j \in \text{KR}(u) : i \notin \text{NN}_k(j) \end{cases}$$

Identically to User  $k$ -NN, there are several similarity measures available. The one chosen for experiments in Chapter 4 is a cosine similarity, which is analogous to the measure introduced in Section 2.2.1.

Another useful assumption Item  $k$ -NN makes is that attractiveness of items don't change quickly over time. This opens up an opportunity to precompute similarities between individual items in training phase and cheaply look them up during recommending, as shown in [11, p. 149–154]. In an online system, it is then sufficient to recompute similarities infrequently (once a day). In the training phase, a matrix  $\mathbf{M} \in \mathbb{R}^{|I| \times |I|}$  is computed, so that

$$\mathbf{M}_{i, j} = \begin{cases} \text{sim}(i, j) & \text{if } i \neq j \wedge \text{sim}(i, j) \in \max_k \mathbf{M}_i \\ 0 & \text{otherwise} \end{cases}$$

Here,  $\max_k \mathbf{M}_i$  denotes a (multi)set of  $k$  largest values among values in  $i$ -th row of  $\mathbf{M}$ .

In order to reduce impact of highly similar items with infrequent interactions, [11] also suggest to normalize rows of  $\mathbf{M}$ , so that  $\forall i \in I : \|M_i\| = 1$ .

For each item  $i \in I$ , matrix  $\mathbf{M}$  therefore contains similarities between  $i$  and  $k$  most similar items different than  $i$ . With  $k \ll |I|$ , the matrix  $\mathbf{M}$  is very sparse, so in practice it takes up a reasonable amount of memory and computations with it are fast. Experiments show that a values of  $k$  in range 5–20 are sufficient.

To score all items for a target user  $u \in U$  at once, there is only one vector-matrix multiplication needed:

$$\text{pred}(u, i) = (\mathbf{M} \cdot u)_i$$

With the precomputation possibility, Item  $k$ -NN overcomes scalability issues for many practical purposes. However, Item  $k$ -NN suffers from cold-start problem to the same extent as User  $k$ -NN, so additional effort must be taken when introducing new items or users to a recommender system.

## Chapter 3

# Design of a Context-Aware Recommender

This chapter introduces proposed modifications of algorithms described in Section 2.2.1 and 2.2.2, which aim to improve recommendations using *context* information. In the field of recommender systems, the context is any additional information carried with an interaction event. The most common context information is a timestamp of interaction, since it is trivial to obtain. Other possible context types include device, screen resolution, OS, geolocation and other data web browsers send, but it may also be obtained from external services (e.g., weather) or explicitly from the user (mood, budget).

Let  $C$  be a tuple of *contextual factors*. Contextual factor is an ordered set of possible *context* values. A contextual factor represents a class of contexts, from which exactly one is effective for each interaction (e.g., contextual factor “season” contains contexts “spring”, “summer”, “autumn” and “winter”).

The prediction function of a context-aware recommender has the form  $\text{pred} : U \times I \times C \rightarrow \mathbb{R}$ , as opposed to  $\text{pred} : U \times I \rightarrow \mathbb{R}$  shown in Section 2.2. In this thesis, tuples in  $C$  always have a length of one, which means that effect of only one contextual factor is studied at the same time. The recommendation space will therefore always be three-dimensional.

### 3.1 Related Research

Research of context-aware recommender systems could be classified into two categories. Some try to segment the multidimensional recommendation matrix into multiple discrete blocks, from which each one corresponds to a different context, and others transform original interactions depending on continuous time dimension. The effect of preferences changing over time is also known as *temporal dynamics*.

The discrete approach is discussed by [12], where a *reduction-based* technique of decreasing the number of context dimensions is introduced. The recommendation space is viewed as an OLAP cube<sup>1</sup>. When recommending for a user in a specific context (e.g. Monday), only data coming from similar contexts (e.g. weekdays) is taken into account. Resulting smaller cube is then flattened into 2D rating matrix using an aggregation function.

An attempt to reduce dimensionality early in the training phase is made by [13]. A portion of items are split into multiple child-items, each of them having only interactions in different context. Standard CF algorithms can then be applied to the resulting rating matrix. This method seems to outperform the reduction-based approach when majority of items are split. Substantial influence of context on user preferences is also indicated by a survey taken in [14].

Temporal dynamics play an important role too, as shown by [15]. It was observed that not only user preferences are variable, but also item popularity change over time.

<sup>1</sup> OLAP is a business intelligence tool for multidimensional data analysis. Learn more from [olap.com](http://olap.com).

At the time of publication, the proposed algorithm, based on SVD++ algorithm<sup>1</sup>, outperformed any previously published results on given dataset.

## 3.2 Proposed Algorithms

This thesis takes ideas from [13] one step further and tries to differentiate the relevance of interactions in different contexts, while not excluding less relevant intractions completely. The temporal dynamics is studied as well and both methods are tested on the same set of datasets in order to be able to compare them mutually.

The common idea, supported by research mentioned in Section 3.1, is that user preferences are dependent on the *target context*, denoting the context in which the user is at the time of the recommendation. Previous interactions of the same user made in different contexts may be less relevant to the recommendation. Furthermore, interactions from different contexts may not even be useful in the model itself, meaning they could be entirely or partially eliminated from the training phase. Both assumptions will be tested individually.

With context, rating matrix described in Section 2.2 is extended by extra dimensions in addition to the original user and item dimensions. Because the traditional CF algorithms are applicable to two-dimensional rating matrices only, dimensions have to be somehow reduced before recommendation. This is done by weighting the interactions according to the similarity of their context to the context of the target user. More detailed description will be provided in later sections.

This thesis studies mainly the time dimension because it can be further partitioned in many ways (month of year, day of week, etc.) and it is often present even in datasets coming from recommenders which aren't context-aware.

### 3.2.1 Temporal Dynamics

The first studied approach to enriching a recommender system with context is temporal dynamics. The hypothesis suggests that older interactions are less relevant to the recommendation than the contemporary ones. There are multiple phenomena supporting this hypothesis:

- Item popularity spikes up because of a fashion trend, or it drops because it is superseded by a newer product.
- User preferences change over time alongside with his age and needs.
- The item has already been bought by the user and items of this sort are not interesting for the user anymore.
- The user searches the item space and is iteratively approaching the desired item—the latest item serves as a better example of the wanted item than the previous ones.

The timeline may be either segmented into time-blocks or be used as an input of a *weighting function*. Segmentation conceptually falls into category of algorithms described in Section 3.2.2, thus it won't be described here.

Weighting function has a signature of  $w : [0, \infty) \rightarrow \mathbb{R}$ . It takes the age of an interaction as an argument and outputs the weight of the interaction. Weighting may be applied either to the rating matrix, to the target user vector, or both. Intuitively, according to scenarios described earlier, weighting of both user vectors and the rating

<sup>1</sup> SVD++ is a matrix factorization algorithm. Learn more from [dl.acm.org/citation.cfm?id=1401944](https://dl.acm.org/citation.cfm?id=1401944)

matrix should have an effect on recommendation. Both cases are tested separately in Chapter 4.

With time dimension, the rating matrix is now 3D<sup>1</sup>. Individual ratings will be denoted  $\mathbf{R}_{u,i,t}$  for user  $u$ , item  $i$  and timestamp  $t$ . Let  $T_{u,i}$  be a set of timestamps of all known interactions of user  $u$  and item  $i$  and let  $t_0$  be the time of recommendation, which is always greater than or equal to any timestamp in  $\mathbf{R}$ . Weighted (2D) rating matrix  $\mathbf{R}'$  is computed the following way:

$$\mathbf{R}'_{u,i} = \begin{cases} \sum_{t \in T_{u,i}} \mathbf{R}_{u,i,t} \cdot w(t_0 - t) & \text{if } |T_{u,i}| > 0 \\ \text{"?"} & \text{if } |T_{u,i}| = 0 \end{cases}$$

Such matrix is ready to be processed by a regular CF algorithm, for which the contextual information and weighting is transparent.

There are several options for choosing the weighting function  $w$ . The requirements for  $w$  are not strict, but this thesis focuses on weighting functions which are decreasing, with the range of values restricted to  $[0, 1]$  with  $w(0) = 1$ . That means the newest interactions have 100 % significance and with increasing age, the significance decreases towards zero.

To control the slope of  $w$ , irrespective of the particular form of the function, a parameter  $\psi$  is introduced, which expresses an analogy of half-life used in nuclear physics. Parameter  $\psi$  expresses a number of days, after which the weight is less than or equal to  $\frac{1}{2}$ . Unless stated otherwise, the unit of  $\psi$  is a day. That puts another constraint on weighting function  $w$ :

$$w(\psi) = \frac{1}{2}$$

*Exponential* weighting function has already been proposed by [16, p. 1264] ( $\lambda$  is a parameter controlling the slope of the function,  $x = t_0 - t$  represents the age of the interaction):

$$w(x) = \exp\left(-\frac{x}{\lambda}\right)$$

When reformulated with  $\psi$ , it gives the first studied weighting function:

$$w_{\text{exp}}(x) = 0.5^{\frac{x}{\psi}}$$

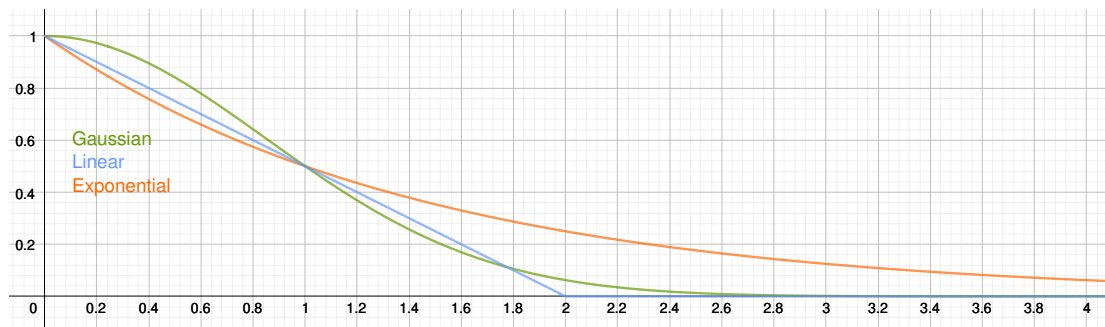
*Gaussian* function is another studied weighting function. Compared to exponential weighting, Gaussian weighting returns higher weights for  $age < \psi$  and nearly zero for  $age > \psi$ , which more distinctly divides interactions between “recent” and “old”. The formula for Gaussian weighting function is:

$$w_{\text{gauss}}(x) = \exp\left(\frac{x^2}{\psi^2} \cdot \ln(0.5)\right)$$

The last examined weighting function is *linear* weighting, which is a drastic alternative to Gaussian weighting, assigning zero to all interactions older than  $2\psi$ . The formula is:

$$w_{\text{linear}}(x) = \max\left\{0, 1 - \frac{x}{2\psi}\right\}$$

<sup>1</sup> Although the object is not a matrix in mathematical sense anymore, it will be referred to as a 3D matrix in order to preserve the established terminology.



**Figure 3.1.** Three different temporal weighting functions for, all with  $\psi = 1$

See Figure 3.1 for visual comparison of all three presented weighting functions  $w_{exp}$ ,  $w_{gauss}$  and  $w_{linear}$ . All three functions will be evaluated when applied to testing user vectors and rating matrix individually, with multiple values of  $\psi$  in range from a few hours to several years. An important observation not visible in the figure is that  $w_{gauss}$  approaches zero significantly faster than  $w_{exp}$  for large  $x$ .

The idea of assessing the interaction importance according to the difference between the recommendation time  $t_0$  and the interaction time  $t$  can be further extended. Most likely, a vast majority of interactions in a dataset will have been made in the distant past, i.e.  $t \ll t_0 - \psi$ . As all three studied weighting functions approach zero quickly for  $t_0 - t > \psi$ , the weights of such interactions will be very small, which can be seen in Figure 3.1. This not only blurs the difference between valuable and unimportant interactions, but the small values may also reach the limits of floating point number representation in the computer.

As an improvement, the global end-of-time value  $t_0$  may be replaced with user-specific values  $t_0(u)$ ,  $u \in U$ , where  $t_0(u)$  denotes the time of the most recent interaction of user  $u$ . Rows of the rating matrix and/or the testing user vectors are weighted using the user-specific  $t_0(u)$  instead of a single global  $t_0$ . The weighting function formulas remain the same, only their argument changes from  $x = t_0 - t$  to  $x = t_0(u) - t$  with  $u$  being the user who owns the interaction. This improved version of temporal weighting will be evaluated for the same set of parameter combinations as the basic version of this algorithm.

### 3.2.2 Discrete Segmentation

Discrete segmentation is another approach to transforming the input 3D rating matrix into a 2D matrix processable with ordinary CF algorithms. It is not limited to time context, but this thesis focuses on it due to data availability. The hypothesis behind segmentation states that when recommending in a specific context, past interactions within the same context are more relevant than interactions made in different contexts.

In the training phase, interactions are accumulated into a multidimensional rating matrix, so that element  $\mathbf{R}_{u,i,c}$  contains a sum of all ratings of user  $u \in U$  and item  $i \in I$  in context  $c \in C$ . The recommender then works with a context-specific rating matrix  $\mathbf{R}_{*,*,c}$  for context  $c$ , which is nothing more than a slice of  $\mathbf{R}$  with the third dimension equal to  $c$ . Analogous slicing can also be applied to a target user vector. Similarly to experiments with temporal dynamics, both variants will be tested both individually and together.

Current research mostly focuses on single context (the target context). For example, when recommending in the “afternoon” context, interactions in “morning” and “night”

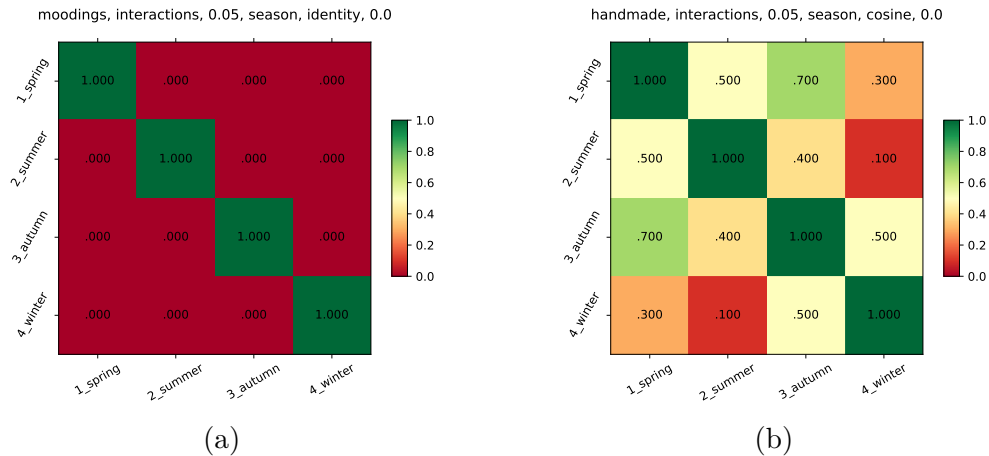
context are completely ignored for the purpose of that particular recommendation. Details of what “ignored” means vary between researchers. Although improvements were observed in articles [12–13], this method has a disadvantage of completely leaving majority of potentially valuable data out.

To fix this flaw, *similarity matrix* is introduced. Similarity matrix  $\mathbf{M} \in \mathbb{R}^{|C| \times |C|}$  is a symmetric matrix whose element  $\mathbf{M}_{c,d}$  represents the degree of similarity between matrices  $\mathbf{R}_{*,*,c}$  and  $\mathbf{R}_{*,*,d}$  for all  $c, d \in C$ . The improved recommender is an ensemble of  $|C|$  ordinary recommenders, each trained with different context-specific rating matrix. When recommending for target user  $u$  in target context  $c$ , items are scored independently by all  $|C|$  recommenders (or more precisely their prediction functions  $\text{pred}_1, \dots, \text{pred}_{|C|}$ ), gaining  $|C|$  partial score predictions for each item. Final recommendation is then computed as a sum of these partial predictions weighted by elements of  $\mathbf{M}$  corresponding to the target context:

$$\text{pred}(u, i) = \sum_{d \in C} \text{pred}_d(u, i) \cdot \mathbf{M}_{c,d}$$

It hasn’t yet been explained how the similarity matrix  $\mathbf{M}$  is created. One option is to make it an identity matrix, i.e.  $\mathbf{M}_{c,d} = \text{diag}(1, 1, \dots, 1)$ . In that case, the algorithm degrades to a single-context recommender, taking only interactions made in the target context into account. For comparison, experiments with identity matrix will be conducted too, but better results are anticipated for more complex similarity matrices.

With knowledge of the recommendation domain, the similarity matrix may be hand-crafted to represent expected similarities between individual contexts. As an example, a handmade similarity matrix for a fictional E-commerce website with clothing is shown in Figure 3.2. The sales are expected to be highly dependent on the outside temperature, which goes hand in hand with the season of the year.



**Figure 3.2.** Examples of a similarity matrix for “season” contextual factor: (a) identity matrix, (b) handmade matrix for a fictional clothing online retailer

As an attempt to deduce similarity matrix from the data itself, a concept of *item-characteristics* of a rating matrix is introduced. Item-characteristics of a rating matrix  $\mathbf{R}$  is a vector  $\text{ichar}(\mathbf{R}) \in \mathbb{R}^{|I|}$  given as

$$\text{ichar}(\mathbf{R})_i = \sum_{u \in U} \mathbf{R}_{u,i}$$

Item-characteristics help us distinguish between best-sellers and marginal items. High value of  $\text{ichar}(\mathbf{R})_i$  means that users often interact with item  $i$ , whereas values close to zero mean users hardly ever interact with it. If an item has a similar value in item-characteristics vectors of two context-specific rating matrices, users interact with the item equally often in both contexts. If many items share a similar value in two contexts (i.e., the whole item-characteristics vectors are similar), users interact with similar items in these contexts, and rating matrix of one context is relevant to the second context, too. This leads to a method of computation of the similarity matrix based on cosine similarity between item-characteristics vectors:

$$\mathbf{M}_{c,d} = \frac{\text{ichar}(\mathbf{R}_{*,*,c}) \cdot \text{ichar}(\mathbf{R}_{*,*,d})}{\|\text{ichar}(\mathbf{R}_{*,*,c})\| \cdot \|\text{ichar}(\mathbf{R}_{*,*,d})\|}$$

This method is expected to naturally capture the internal structure of the dataset. For this reason, similarity matrix constructed via item-characteristics is also a convenient tool for visualisation of periodicity, or generally any context-specificity in the dataset. Examples of real similarity matrices obtained from experimental data will be shown in Chapter 4 along with other properties of the testing datasets.

Since for some datasets and contextual factors, differences between contexts may be very small, it may be preferable to scale the values inside the similarity matrix, so that the smallest value becomes zero, the largest becomes one, and the values between scale linearly into  $[0, 1]$  interval:

$$\forall c, d \in C : \mathbf{M}'_{c,d} = \frac{\mathbf{M}_{c,d} - \min \mathbf{M}}{\max \mathbf{M} - \min \mathbf{M}}$$

This adjustment magnifies the differences while relatively preserving the original structure. Experiments involving similarity matrix will use the identity matrix, the similarity matrix based on item-characteristics, and its variant scaled to  $[0, 1]$  interval.

### 3.3 Evaluation Framework

For measuring the performance of the proposed algorithms, an evaluation framework has been designed and implemented. Details of the evaluation methodology are described in following section. Details about the usage of the framework can be found in Appendix A.

The framework follows the typical machine learning evaluation pipeline. The data is split into two groups—*training* and *testing*. The training set contains the bigger ratio of the available users and the testing set contains the rest. The training set forms a rating matrix, or a set of context-specific matrices in case of the segmentation algorithm.

As described in Section 3.2.2, when using the segmentation method, the rating matrix may be sliced into multiple context-specific matrices. On each context-specific slice of the rating matrix, a separate model is trained (for temporal dynamics algorithm, there is only one rating matrix, thus only one model is trained). The architecture of the framework allows to plug multiple algorithms in, so both User  $k$ -NN and Item  $k$ -NN may be evaluated using the same common preprocessing and postprocessing steps.

The testing set contains the testing user vectors. A typical approach to the testing phase is *leave-one-out* method, which sequentially “hides” one interaction from the user vector and feeds the trained model with the remaining interactions of the user. The model then predicts top- $n$  items for this user. If the item of the hidden interaction



is contained in the predicted top- $n$  list, the prediction is considered successful, as the hidden interaction has truly been performed by the user. This procedure is repeated for each testing user.

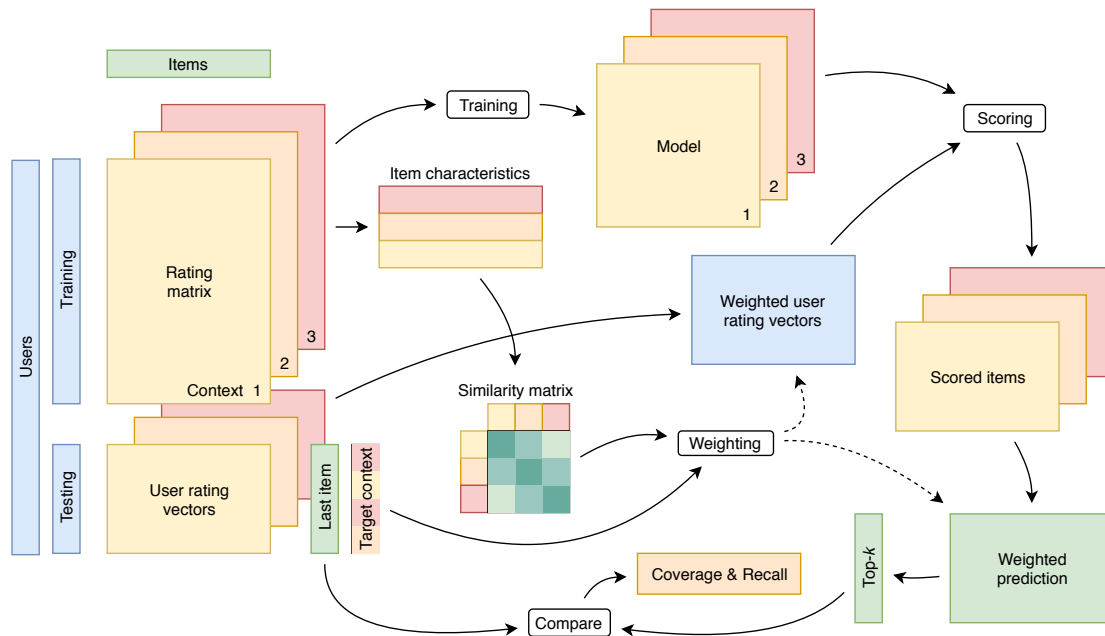
As the proposed algorithm heavily exploits the time dimension of the data, leave-one-out seems insufficient, because it demands predictions for interactions preceding the interactions available to the recommender as an input. To maintain correct real-life causality, a slight modification of this method, *leave-last-out*, is used. This method hides only the most recent interaction, thus all past interactions are available to the model when making a prediction. The last interaction of each testing user also determines the target context.

When using the segmentation algorithm, the item-characteristics and a similarity matrix are derived from the rating matrix. If the user vectors are split into several context-specific parts, the vectors are first merged together using the weighting procedure described in Section 3.2.2. For each testing user, a prediction is then generated in the leave-last-out fashion using each trained model, resulting in a set of vectors of scored items for each testing user. These predictions are then merged together using the same weighting process as the user vectors.

Finally, the resulting predictions are compared to the true last interactions of the testing users, and performance metrics are computed. The choice and properties of the selected performance metrics are covered in Section 4.1.

The schema of the framework is outlined in Figure 3.3. The schema illustrates the most complex variant, when both the rating matrix and the testing user vectors are partitioned into context-specific segments. When only one or none of the two is segmented, the principles of the function remain the same, only the number of models / user vector sets is 1 and the corresponding weighting step is redundant.

If only user vectors are to be segmented, the similarity matrix is computed from the segmented rating matrix, but the (single) model is trained from a single, compound rating matrix.



**Figure 3.3.** Schema of the evaluation framework

# Chapter 4

## Experiments

An extensive set of experiments have been conducted in order to verify the hypotheses stated in Chapter 3. All context-aware versions of collaborative filtering algorithms described in Section 3.2 are evaluated using the framework designed in Section 3.3 on four datasets, which will be described in Section 4.2. Both User  $k$ -NN and Item  $k$ -NN are used as the underlying base algorithms.

Over 12 500 different combinations of algorithm variants / parameters were computed using a power of more than 12 cpu-months. Only a fragment of the most interesting results are shown here, others are only briefly summarized. The file with the full set of measured data is attached to this thesis, see Appendix A.

### 4.1 Performance Metrics

There is no single consensus on performance metric for measuring success rate of a recommender system. However, some are suited better for implicit datasets. Performance metrics can be categorized into two groups—*statistical accuracy* metrics and *decision-support accuracy* metrics. Examples of statistical accuracy are root mean squared error (RMSE) or mean absolute error (MAE) and they measure the difference between expected rating of an item and its actual rating in a certain way. Due to this nature, they are better suited for datasets with explicit ratings with broader range of values.

For datasets with implicit ratings and the top- $n$  recommendation, which is the case of datasets examined in this thesis, decision-support accuracy metrics are more suitable [17, p. 229]. Examples of this kind of metrics is *precision*, *recall*, *F-measure* or *ROC curve*. They focus on presence or absence of the expected item in the recommended top- $n$  list, regardless of the precise item ranks.

The most popular options seem to be precision and recall [18, p. 226]. Furthermore, for implicit ratings, these two metrics behave similarly. Recall has been chosen for experiments in Chapter 4.

A general evaluation pattern of the leave-last-out method is that for each one of the  $m$  testing users, their last interaction is hidden. A model predicts top- $n$  items to each testing user, yielding  $m$  lists of  $n$  items. For recall, the ordering inside the lists is not important, so let  $\text{top}_u$  denote a set of  $n$  items recommended for user  $u \in \{1, \dots, m\}$ . The item of the hidden interaction of each user is then compared to the corresponding top- $n$  list. Let  $r$  be the number of users, for whom their hidden item is found in the top- $n$  list, meaning the recommender recommended a relevant item to the user.

Recall is computed as a ratio of the number of relevant recommended items to the number of all relevant items, which in the case of leave-last-out top- $n$  recommendation translates into following formula:

$$\text{recall} = \frac{r}{m}$$

In addition, *catalog coverage* has been chosen as a secondary metric. Catalog coverage measures the variety and richness of recommendation, which is a desired property of

a recommender system [19, p. 40]. A recommender with high catalog coverage allows users to discover a broad range of new content. It is computed as a ratio of the number of unique recommended items to the number of all known items, which is

$$\text{catalog coverage} = \frac{|\bigcup_{u \in \{1, \dots, m\}} \text{top}_u|}{|I|}$$

## 4.2 Testing Datasets

The datasets chosen for the experiments are collections of data gathered from real online retailers and content providers. They contain nothing but implicit interactions. Each dataset contains data provided by a different company, offering a different kind of products, so the influence of context is expected to vary between datasets.

The datasets have been preprocessed to include only interactions of users who had performed at least two interactions. Users with one interaction (or none) don't contribute to the quality of the model and they would only distort the results. Approximately 10% of interactions have been eliminated in this way.

The websites from which the datasets originate have already been using a recommender system, thus the data may be slightly biased, because the user's choice is influenced by the recommendations (which, in fact, is the purpose of recommender systems). In studied datasets, about 5–15% of all user interactions have been recommended by the existing recommender, the rest are spontaneous interactions.

The websites target users within a single time zone, so there is a clear daytime periodicity. Afternoon and evening, which are the most frequent parts of day, receive about 4× to 10× more traffic than morning times.

The basic properties specific to the individual datasets follow.

### Furniture

This dataset contains design furniture and interior equipment of various types, from couches and tables to lamps and flowerpots. No specific context influence is expected a priori.

### Posters

This dataset contains posters, usable as an indoor decoration. There are no other kinds of items, thus even less contextual diversity than in Furniture dataset is expected.

### Outdoor

Dataset with outdoor clothing and equipment. Popularity of such items is expected to be highly dependent on weather, therefore season (part of the year) may play an important role for recommending.

### Streaming

Multimedial dataset, which doesn't originate from a online retail but rather from a content provider, more specifically a video streaming platform. This dataset is expected to be influenced by time of day to a larger extent than usually; day of week may also have some impact. A speciality of this dataset is that it is highly volatile over time, and partitioning the time dimension into longer periods of time (e.g. month of a year) doesn't make sense, because most items are replaced faster than that.

Sizes of the rating matrices for these four datasets are presented in Table 4.1.

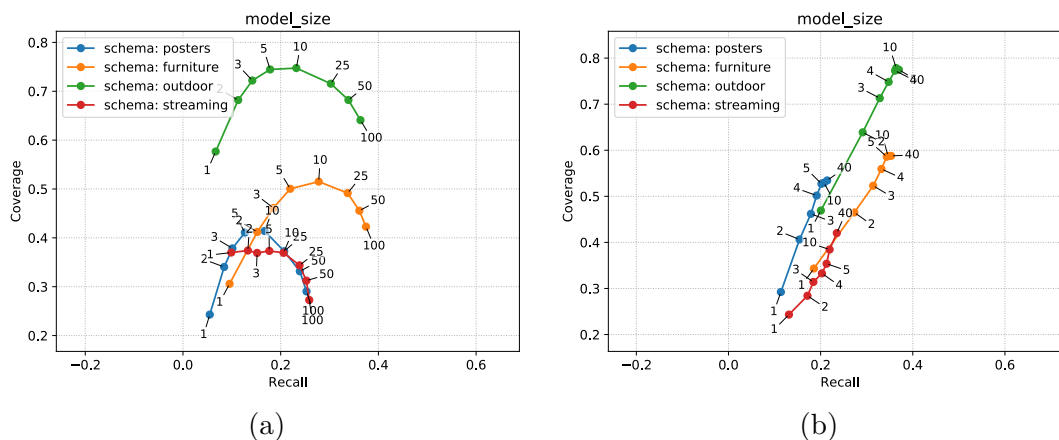
Dataset	Users	Items	Ratings	Density
Furniture	190 000	12 000	1 270 000	0.059 %
Posters	54 000	3 600	256 000	0.137 %
Outdoor	550 000	7 700	3 120 000	0.077 %
Streaming	550 000	10 000	31 400 000	0.586 %

**Table 4.1.** Rating matrix dimensions of the experimental datasets, where column “User” corresponds to  $|U|$ , “Item” to  $|I|$ , “Ratings” stands for the number of known interactions and “Density” expresses the ratio of known interactions to the overall number of elements in the rating matrix

### 4.3 Fixed Parameters

For both User  $k$ -NN and Item  $k$ -NN, an appropriate model size  $k$  must be chosen. With too small values of  $k$ , the model overfits and for large  $k$ , memory and computational complexity increases.

Figure 4.1 shows the impact of manipulating model size on performance of User  $k$ -NN and Item  $k$ -NN in Recall-Coverage plane. In case of User  $k$ -NN, there is no clear “best” value of  $k$ . We can see that about  $k = 10$  is the first Pareto-optimal with the highest coverage, but increasing  $k$  leads to better recall even for high values of  $k$ . Due to this ambiguity, experiments are performed for multiple values of  $k$ , namely 10, 25 and 100. Nevertheless, the results are usually consistent for multiple model sizes, so for clarity of the diagrams, usually only User 25-NN is shown.



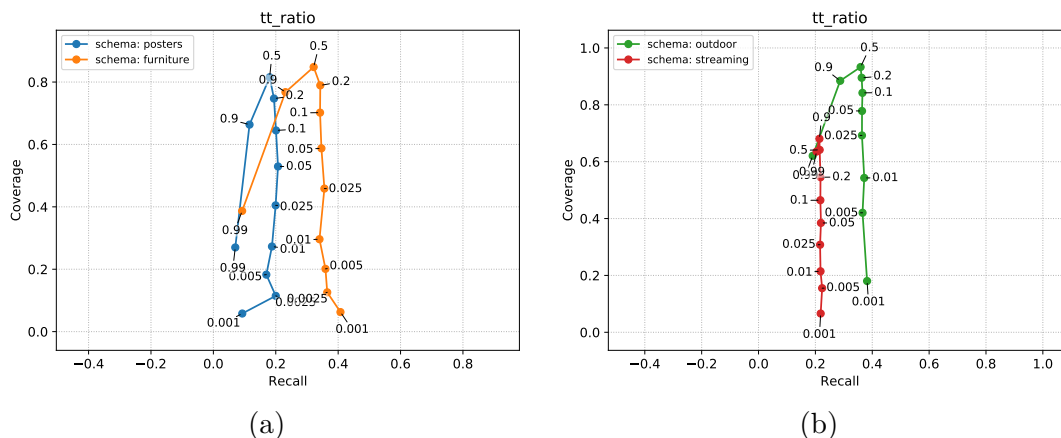
**Figure 4.1.** Impact of model size to recall/coverage for (a) User  $k$ -NN, (b) Item  $k$ -NN

The situation is different for Item  $k$ -NN. Both recall and catalog coverage grows steadily until  $k$  reaches the number of recommended items, which is 5. From that point, increasing  $k$  has minimal effect on recall and coverage. This is caused by the fact that for each item, there is an item which has the greatest similarity score to the first item (it is the most similar item). As the items with which the target user has interacted in the past vote for the final recommendation, the most similar items usually get into the top- $n$  list. For  $k > n$ , the items beyond  $n$  simply have too low voting weight to score their most similar item high enough to get into the top- $n$  list. However, to avoid overfitting of the Item  $k$ -NN algorithm, the value of  $k$  is fixed to 10.

Another fixed parameter is the ratio of testing and training set size. Typical testing set size used in machine learning is around 20 %, but since the testing datasets used

for experiments are fairly large and the testing phase is computationally expensive, the value of 10% is used for Posters dataset and 5% for the remaining three (they are larger and don't require that large testing set).

Figure 4.2 shows the measured performance of unmodified Item  $k$ -NN algorithm with different testing set size. For the smaller datasets, very low testing/training set size ratio leads to unstable results, whereas large testing set size leads to degraded performance, because the training set is too small. Large datasets are more robust against manipulation of the ratio, because even the extreme values leave enough data in both sets. Figure 4.2 suggests that the chosen values of testing/training ratio are sufficient.



**Figure 4.2.** Impact of testing/training set size ratio on measured recall and coverage of unmodified Item  $k$ -NN algorithm

The value  $n$  of the top- $n$  list of recommended items to each user is set to be 5 for all experiments. This is the usual number of items recommended in online shopping, so it conforms to the real usage of recommender systems.

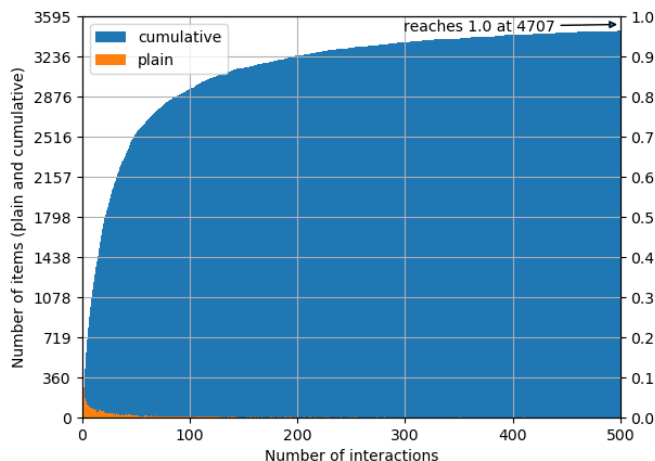
## 4.4 Item Significance

The distribution of best-sellers versus long-tail items is nearly exponential, as can be seen in Figure 4.3 of Posters dataset. Distribution in other datasets is similar. In order to reduce noise in the data when computing the similarity matrix, one option is to take only a certain percentage  $\alpha \in (0, 1]$  of the most popular items into consideration when constructing the item characteristics vectors.

Both conservative values  $\alpha \in \{0.99, 0.95, 0.9\}$  and more drastic values  $\alpha \in \{0.5, 0.1, 0.05\}$  had led to negligible change in recall/coverage, thus this modification is not used in further experiments.

## 4.5 Temporal Dynamics

First, the variant with the global  $t_0$  end-of-time is discussed. In the second part, results of the algorithm with times  $t_0(u)$  for individual users are shown. Results are presented in a form of recall-coverage diagram with recall on  $x$ -axis and catalog coverage on  $y$ -axis. The iterated variable is stated in the legend of the diagram. Reference values of plain



**Figure 4.3.** Distribution of the items in Poster dataset based on their popularity (histogram shown in orange, the corresponding empirical distribution function in blue)

User  $k$ -NN and Item  $k$ -NN algorithms are always drawn in grey color. For Item  $k$ -NN, the reference value is for  $k = 10$ . For User  $k$ -NN, either a point with a specific value of  $k$  is drawn, or a curve is drawn for multiple values of  $k \in [3, 150]$  due to reasons noted in Section 4.3. For the sake of readability, some datapoints were purposely omitted, and for User  $k$ -NN, usually only  $k = 25$  curves are shown. The curves for  $k = 10$  and  $k = 100$  follow the shape of  $k = 25$ , they are just shifted in the coverage-recall space.

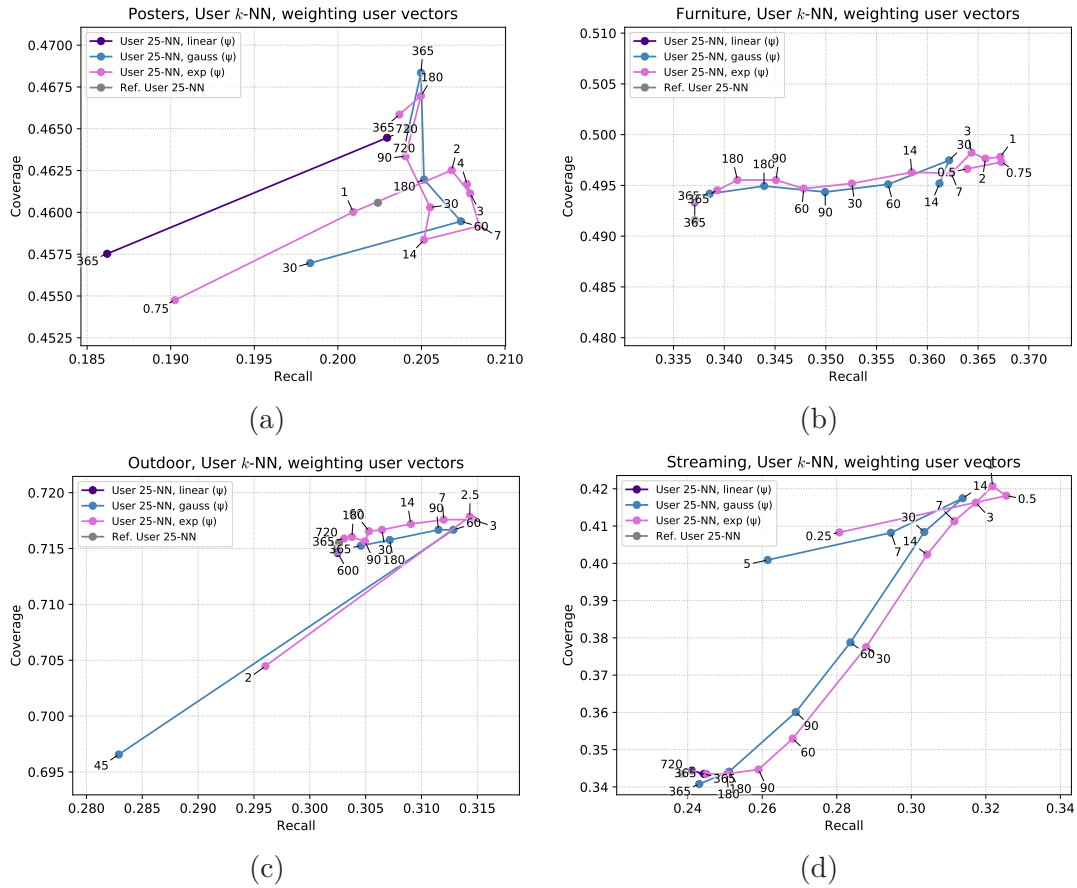
#### 4.5.1 Weighting the User Vectors by Global End-of-Time

The first set of experiments keeps the rating matrix intact and the weighting is applied only to the testing user vectors. Weighting of the rating matrix is studied in other sections. This set of experiments uses a global time of recommendation  $t_0$ . That's to say, the weight of an interaction made in time  $t$  is computed as  $weight = w(t_0 - t)$ ,  $t_0$  being a dataset-specific constant.

When comparing weighting functions  $w_{\text{exp}}$ ,  $w_{\text{gauss}}$  and  $w_{\text{linear}}$ , it can be seen that both the exponential and Gaussian weighting achieve comparable results (see Figures 4.4 and 4.5). However, the Gaussian function requires more data (larger  $\psi$ ) to be able to compete with  $w_{\text{exp}}$ , because for  $t \ll t_0 - \psi$  (which is the case for majority of interactions),  $w_{\text{gauss}}$  assigns much smaller weights than  $w_{\text{exp}}$ . No improvement is observed for  $w_{\text{linear}}$  for similar reasons. Linear weighting assigns zero weight to most interactions, so the recommender has insufficient amount of nonzero data left to be able to predict sensible items.

Only the tips of  $w_{\text{linear}}$  lines are shown as they rapidly fall down to recall = 0 for  $\psi < 365$ , which would obscure the improvements introduced by  $w_{\text{exp}}$  and  $w_{\text{gauss}}$  if shown in the same diagram.

A significant improvement can be observed on Furniture, Outdoor and Streaming datasets using this technique. It's not a surprise that the best results were achieved on Streaming dataset, where the items are replaced often and users never interact with old items (these are removed from the website, but the recommender keeps track of them). Recall has been increased from 0.238 to 0.326 for User  $k$ -NN with exponential weighting function and  $\psi = 0.5$ , which is a relative 37% improvement (see Figure 4.4 d). A moderate improvement can also be observed in Furniture and Outdoor datasets, with recall gain of 9% and 4% respectively. For Posters dataset, the results are too



**Figure 4.4.** Different User  $k$ -NN weighting functions applied to user vectors of datasets (a) Posters, (b) Furniture, (c) Outdoor, (d) Streaming, with  $t_0$

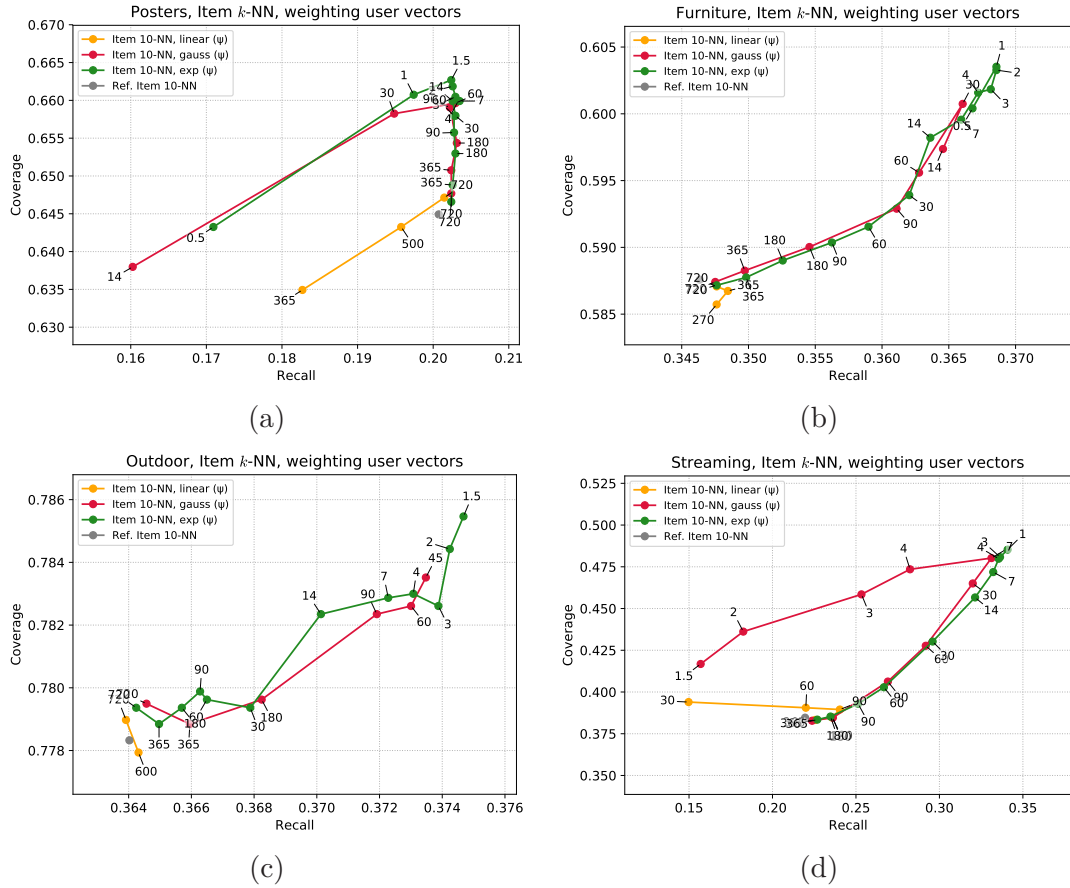
inconsistent to be relevant, which is possibly caused due to a rather small size of this dataset. The weighting technique behaves similarly well when using Item  $k$ -NN as its base algorithm (Figure 4.5).

The trend of both exponential and Gaussian weighting recall-coverage curve is the following. Starting with the unmodified algorithm, which corresponds to  $\psi = \infty$ , the value of  $\psi$  is decreased and the old interactions are being forgotten from the user’s vector. This leads to an increase in recall, which supports the hypothesis that older interactions are less relevant than the recent ones. At the same time, coverage increases, too, because user preferences are more diverse when looking only at the recent interactions. When  $\psi$  decreases to values of around 1 day, too much information is being near-zeroed and the recommendation quality drops quickly. These results show that interactions older than  $\sim 2$  days don’t contribute much to the recommendation.

## 4.5.2 Weighting the Rating Matrix by Global End-of-Time

Interestingly, weighting the interactions in the rating matrix results in no recall improvement at all. The results are shown in Figure 4.6 for exponential weighting function only, because it performed best in Section 4.5.1.

The reason why weighting of the user vectors improves recall and weighting of the rating matrix makes it worse is probably the fact that although the old interactions are irrelevant for extraction of users’ preferences, the relations captured by the rating matrix



**Figure 4.5.** Different Item  $k$ -NN weighting functions applied to user vectors of datasets (a) Posters, (b) Furniture, (c) Outdoor, (d) Streaming, with  $t_0$

are steady over time. Despite the user preferences are dynamic, the old interactions are still valuable for computing user-user or item-item similarities in nearest neighbor algorithms.

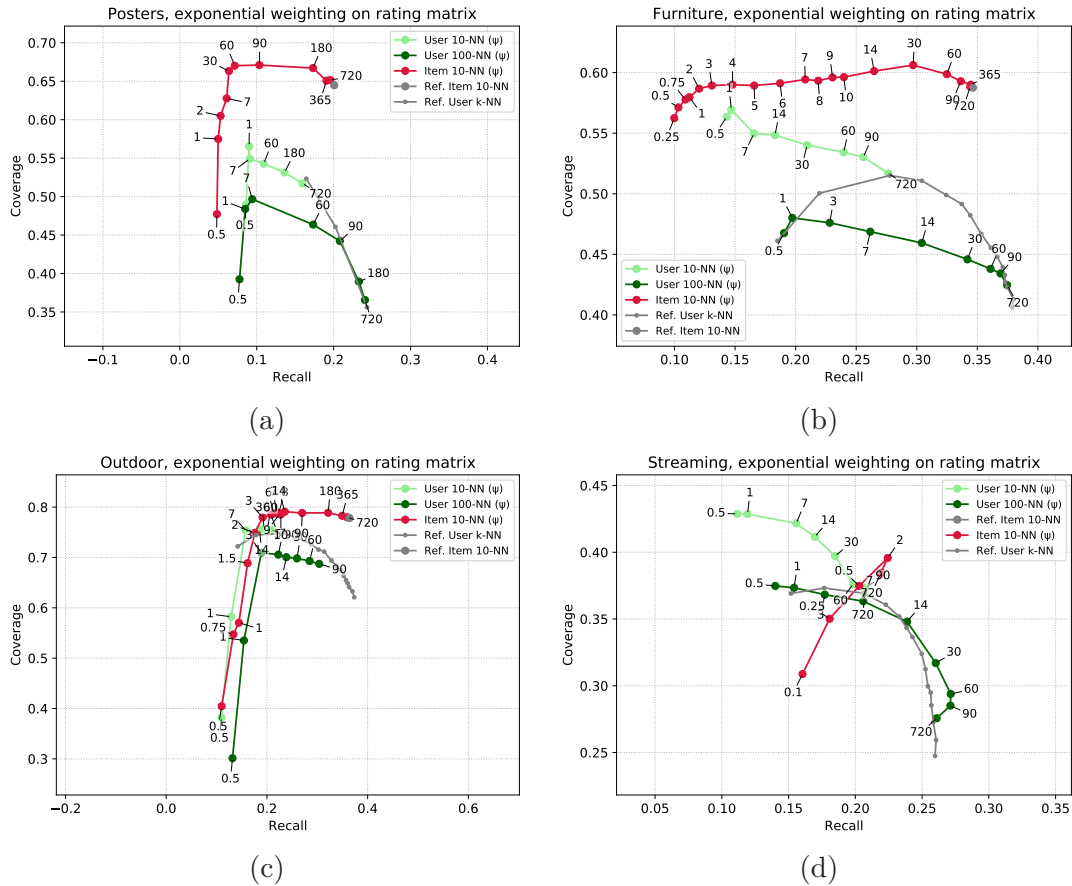
### 4.5.3 Weighting the User Vectors by User-Specific End-of-Time

Contrary to experiments described in Section 4.5.1 and 4.5.2, the experiments described in this section use user-specific time of recommendation. Instead of having a constant, dataset-specific parameter  $t_0$ , which is used as a part of the input to the weighting function, a set of  $|U|$  parameters  $t_0(u_1), t_0(u_2), \dots, t_0(u_{|U|})$  is used, one for each user. The value of  $t_0(u)$  is the time of the most recent interaction of user  $u$ . When either building the rating matrix or weighting the user vectors, the weight of the interaction performed in time  $t$  by user  $u$  given a weighting function  $w$  is computed as

$$weight = w(t_0(u) - t)$$

Compared to the previous algorithms with a global  $t_0$  parameter, the algorithm with user-specific last interaction times should more accurately express the dynamics of user preferences. Rather than capturing the overall dynamics of all known users, computing the age of the interaction as  $age = t_0(u) - t$  should help to differentiate between recent and old interactions of given user, even if all user's interactions are relatively old with respect to the time of recommendation. Therefore, better results are expected than those achieved in Section 4.5.1.





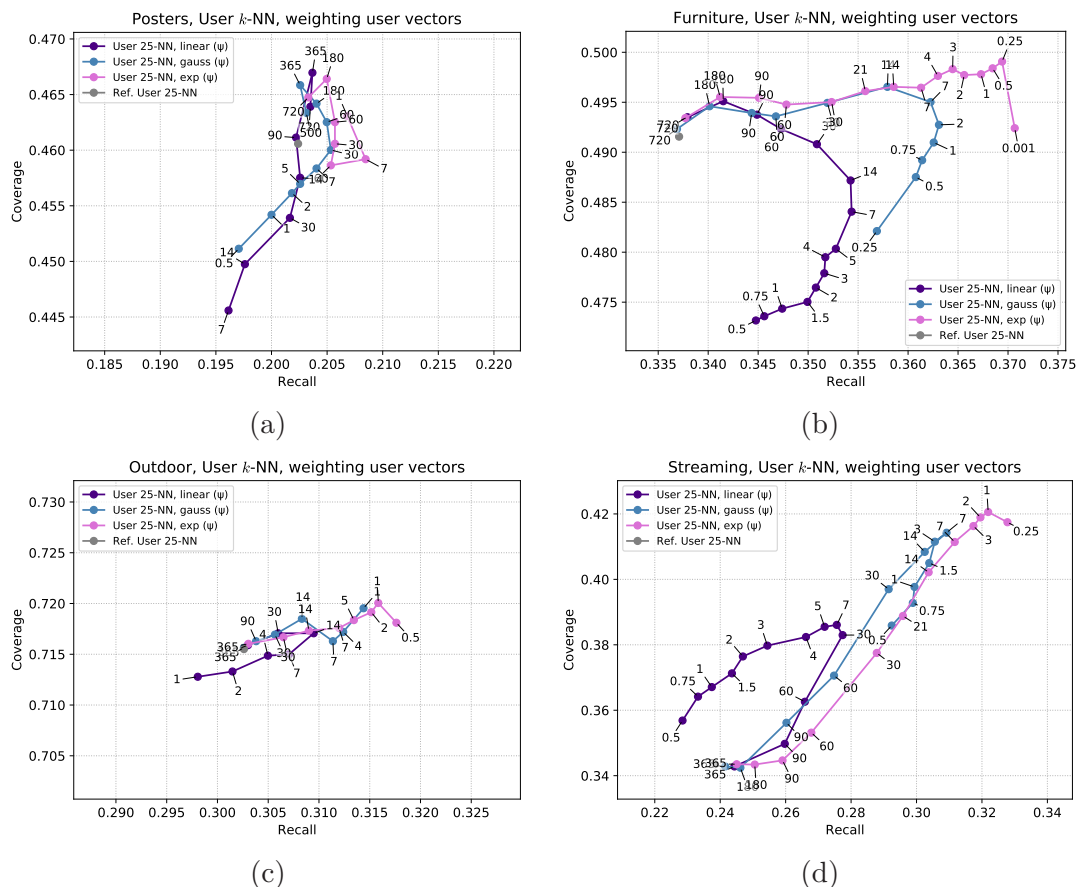
**Figure 4.6.** Exponential weighting applied to rating matrix of datasets (a) Posters, (b) Furniture, (c) Outdoor, (d) Streaming, with  $t_0$

The measured values for User  $k$ -NN are shown in Figure 4.7 and for Item  $k$ -NN in Figure 4.8.

There are two main differences between the results of this set of experiments and the results obtained with a global  $t_0$  parameter. First, the linear weighting function  $w_{\text{linear}}$  catches up with  $w_{\text{gauss}}$  and  $w_{\text{exp}}$ , even if it still performs the worst of these three. Unlike the case with single global  $t_0$ , where it zeroes most of the interactions out, the version with user-specific  $t_0(u)$  leaves a few nonzero interactions for every user. The recommender has enough information to be able to compute reasonable recommendations. The fact that the linear weighting function performs worse than weighting functions which are above zero even for large argument indicates that it is not wise to discard old interactions completely, even if lowering their importance has a positive impact on recall.

Second, the point when the advantages of lowering the weight of old, irrelevant interactions are balanced out with the disadvantages of forgetting too much information occurs for lower values of  $\psi$ , approximately  $\psi = 7$  (as opposed to 14–60 seen in Section 4.5.1), which is natural because the users whose all interactions are old need larger time span when using a global  $t_0$ .

The other characteristics of the measured data are similar to those obtained in previous sections, only the results for Posters dataset are more consistent and a small improvement over the reference algorithm is visible. Recall is the highest with the ex-



**Figure 4.7.** Different User  $k$ -NN weighting functions applied to user vectors of datasets (a) Posters, (b) Furniture, (c) Outdoor, (d) Streaming, with  $t_0(u)$

ponential weighting function, just as in Section 4.5.1. However, slightly better results were obtained using user-specific  $t_0(u)$  rather than a single  $t_0$ . The relative increase in recall is 10% for Furniture dataset, 5% for Outdoor, 38% for Streaming and 3% for Posters dataset.

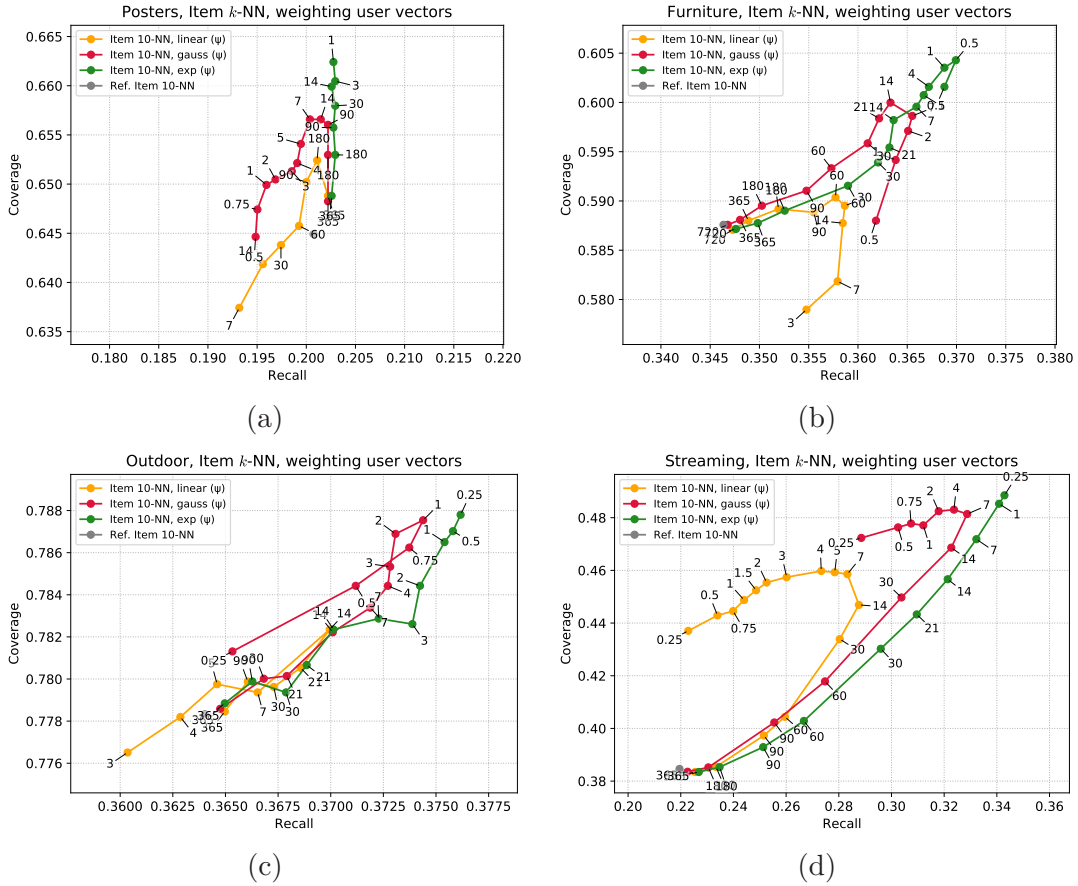
#### 4.5.4 Weighting the Rating Matrix by User-Specific End-of-Time

Similarly to global end-of-time discussed in Section 4.5.2, when weighting the interactions in the rating matrix and using User  $k$ -NN as a base algorithm, coverage increases and recall decreases (after a short period of insignificant increase for User 100-NN) as shown in Figure 4.9.

Nevertheless, the situation is different with Item  $k$ -NN (Figure 4.10). Weighting the rating matrix with individual  $t_0(u)$  actually improves the recall. For Posters and Furniture datasets, this method even reaches the highest recall among all temporal dynamics experiments conducted for purposes of this thesis, the recall improvement is 11% and 12% respectively.

In all previous experiments, User  $k$ -NN and Item  $k$ -NN responded similarly to the proposed enhancements. However, weighting the rating matrix using a user-specific  $t_0(u)$  parameter leads to a huge difference between the behaviour of the two base algorithms.

The reason why Item  $k$ -NN works better with weighting and User  $k$ -NN fails is probably that when a large portion of interactions are weighted close to zero, the

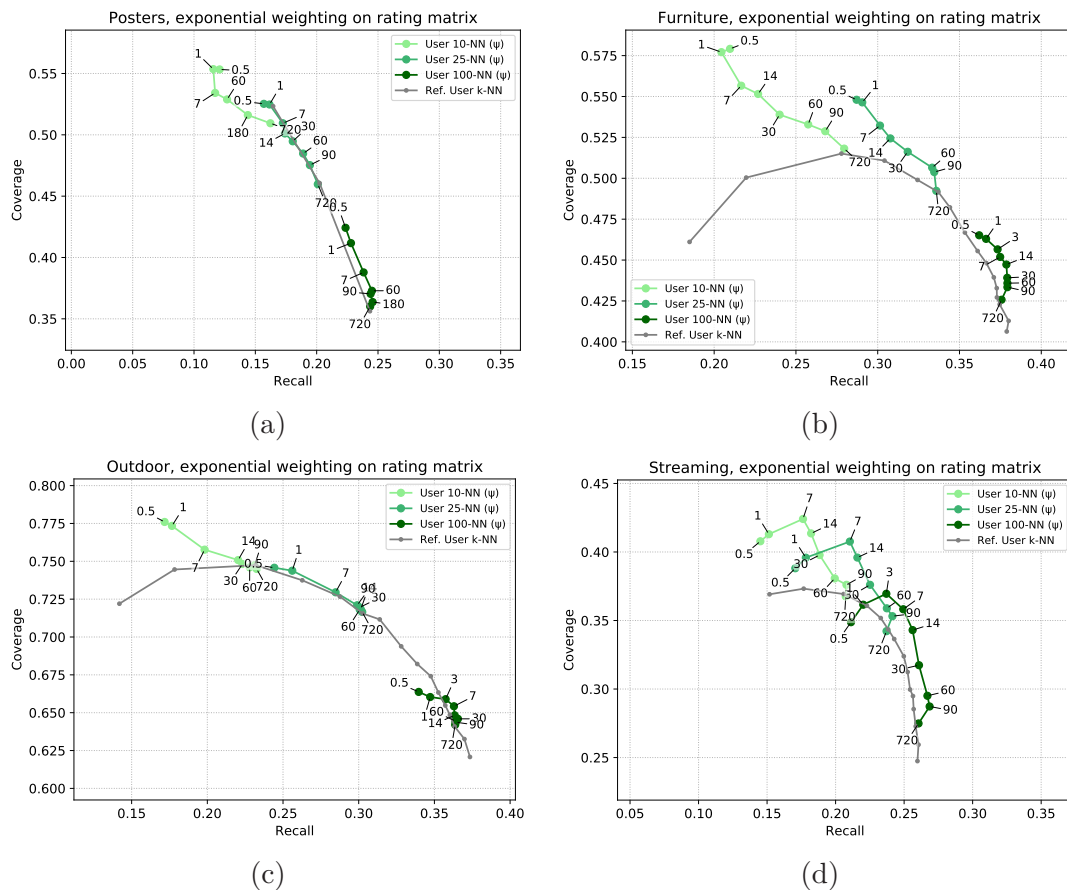


**Figure 4.8.** Different Item  $k$ -NN weighting functions applied to user vectors of datasets (a) Posters, (b) Furniture, (c) Outdoor, (d) Streaming, with  $t_0(u)$

differences between users are increasing (there is a lower probability of overlapping items) and it's harder to find the set of  $k$  most similar users to the target user. On the other hand, Item  $k$ -NN is more robust to weighting. Items usually have more interactions than users since  $|U| > |I|$  in most datasets. Item  $k$ -NN is able to maintain the list of  $k$  most similar items to the target item even when the interaction values are lowered by the weighting.

This positive result hasn't appeared with  $t_0$  parameter because most interactions were made in time  $t \gg \psi$ , meaning most weights are close to zero and the similarities between items got blurred, lowering the quality of recommendation. This didn't happen when using user-specific  $t_0(u)$ , because each user owns a few high-weighted interactions and the relations between items remain strong enough.

This observation partially contradicts the results of experiments in Section 4.5.2, which can be now more accurately interpreted. It shows that even the mutual similarities between items develop over time, but focusing on the recent interactions is beneficial only in case enough data with high weights is present.



**Figure 4.9.** User  $k$ -NN with exponential weighting applied to the rating matrix of datasets (a) Posters, (b) Furniture, (c) Outdoor, (d) Streaming, with  $t_0(u)$

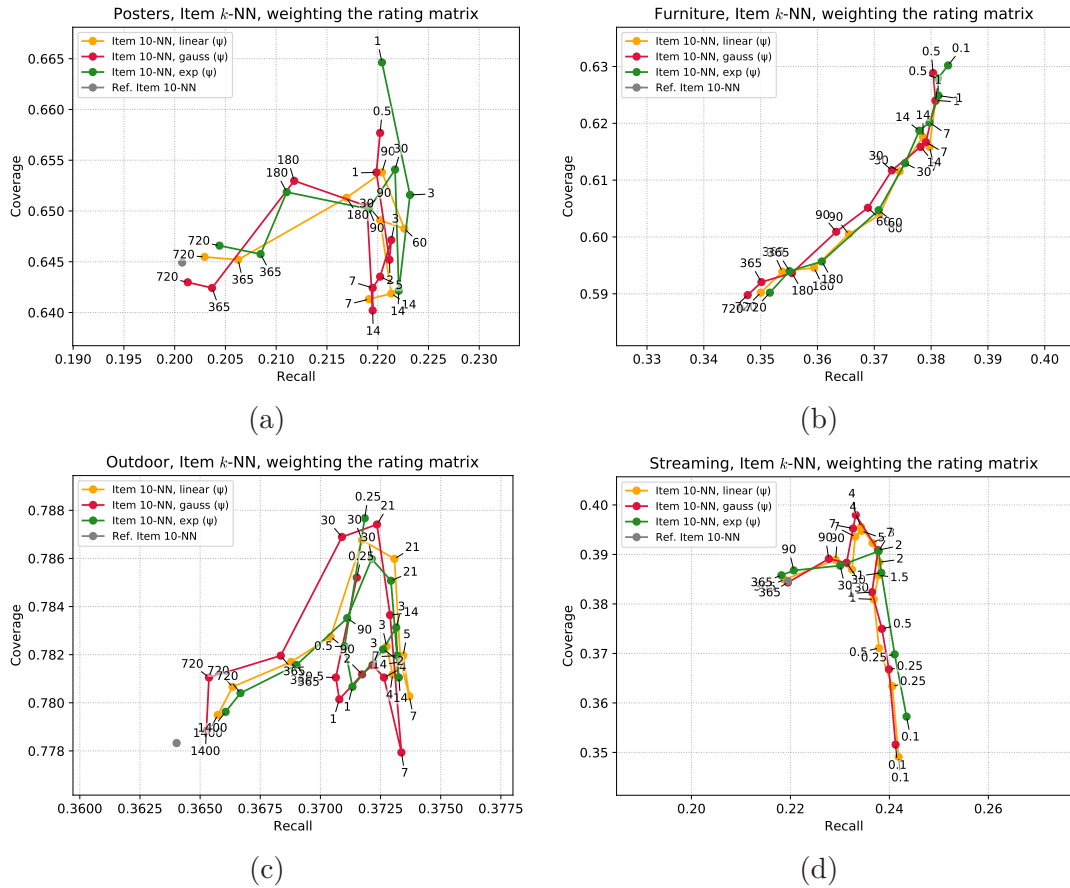
## 4.6 Discrete Segmentation

This section describes the results of experiments with algorithm variations introduced in Section 3.2.2. The input data is categorized into multiple classes, leading to either multiple rating matrices and, in consequence, multiple models, which will vote for the final prediction, or multiple sets of user vectors, which will be evaluated individually and the final prediction will be computed as a weighted sum of these partial predictions.

The weighting coefficients are implemented by a similarity matrix  $\mathbf{M}$ . In this section, recall and catalog coverage is measured for different choices of  $\mathbf{M}$  corresponding to different contextual factors:

- Day of week – seven categories, from Monday to Sunday.
- Season – Spring, Summer, Autumn and Winter, each three months long, spring starting on 1st March.
- Month – the year is sliced more finely than into seasons.
- Daytime – five unequally long periods of time (morning from 5.00 to 9.00, forenoon from 9.00 to 13.00, afternoon from 13.00 to 18.00, evening from 18.00 to 13.00, night from 13.00 to 5.00).
- Hour – finer partitioning of the day into individual hours.

To be able to compare the performance of the modified algorithms, catalog coverage and recall of the unmodified User 25-NN and Item 10-NN algorithms are stated in



**Figure 4.10.** Different Item  $k$ -NN weighting functions applied to the rating matrix of datasets (a) Posters, (b) Furniture, (c) Outdoor, (d) Streaming, with  $t_0(u)$

Dataset	User 25-NN		Item 10-NN	
	Coverage	Recall	Coverage	Recall
Furniture	0.49156	0.33709	0.58758	0.34635
Posters	0.46058	0.20239	0.64492	0.20074
Outdoor	0.71551	0.30261	0.77833	0.36402
Streaming	0.34350	0.23838	0.38462	0.21953

**Table 4.2.** Reference coverage and recall for User 25-NN and Item 10-NN

Table 4.2. Recall and coverage of the contextual algorithms will be compared to those reference values.

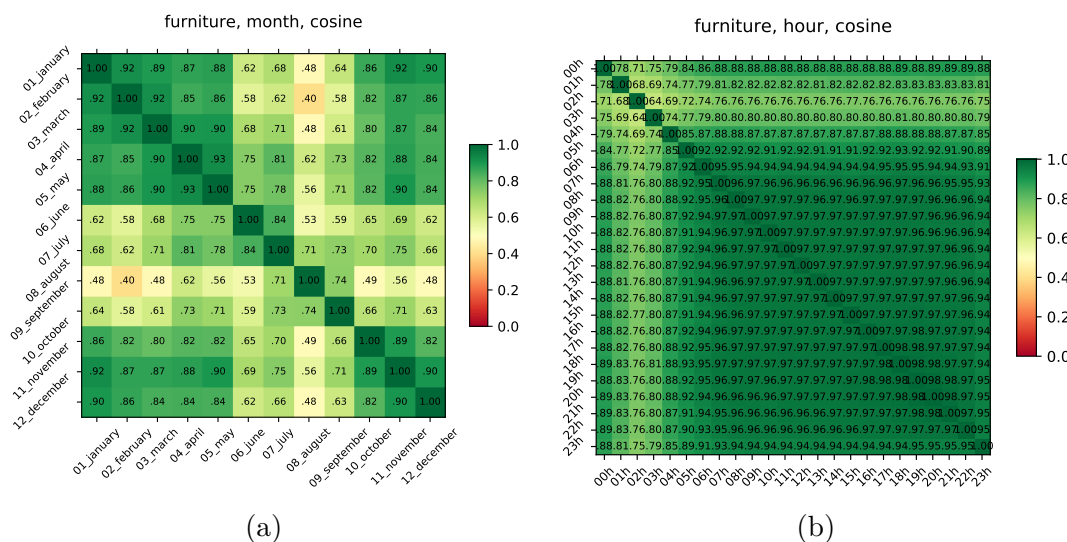
This section will be further divided into subsections corresponding to the individual datasets, because the appearance of the similarity matrices—core of the discrete segmentation methods—are highly dependent on the internal periodicity of the dataset.

It turns out that partitioning the testing user vectors into multiple context-specific doesn't improve recall. Since majority of the users in all examined datasets have been active only for a short period of time, all of the the user's interactions belong to a single context and the vectors corresponding to other contexts are zero vectors. The context-aware recommender then degrades to a basic collaborative filtering recommender. For this reason, the presented results are results of experiments with multiple context-

specific rating matrices but a single set of testing user vectors (unless explicitly specified otherwise).

#### 4.6.1 Furniture Dataset

Although no strong periodicity had been predicted for Furniture dataset, as the furniture sales are intuitively independent of weather conditions or part of the day, two relatively strong tendencies have been discovered for month and hour contextual factors. The similarity matrices, expressing similarities between all pairs of contexts belonging to a given contextual factor, are shown in Figure 4.11. It can be clearly seen that different items are being bought from October to May than in the Summer period in Figure 4.11 a. Figure 4.11 b shows that items bought at night differ from items bought during the day. However, this may be simply an effect of not having enough night interactions. The retailer is Europe-based and most people don't order furniture from midnight to dawn, causing random items to appear in this night period, which have low cosine similarity to the ordinary item popularity.



**Figure 4.11.** Similarity matrices of natural periodicities in Furniture dataset computed using cosine similarity between item-characteristics of (a) month, (b) hour of a day

Similarity matrices for other contextual factors are close to all-one matrices (i.e.,  $\forall i, j \in \{1, \dots, |I|\} : \mathbf{M}_{i,j} = 1$ ), which means most of the items are purchased equally often. To reveal the indistinctive periodicity even for those contextual factors, scaling to  $[0, 1]$  interval had been applied to these similarity matrices, as mentioned in Section 3.2.2. For Furniture dataset, the scaling doesn't improve recall much, which is a consequence of the scaling being an artificial distinguishing tool, while apparently the plain cosine-based similarity matrix represents the relations between contexts accurately.

Measured recall and catalog coverage for all studied contextual factors can be seen in Table 4.3. Recall of models built on the similarity matrices shown in Figure 4.11 are highlighted and they present the largest recall improvement among the measurements (14% for User 25-NN with per-hour segmentation).

The identity matrix decreases recall in all cases, meaning the improvement caused by focusing on the target context don't compensate for the loss of the training data (the only model which has a nonzero voting weight in the ensemble is the one trained exclusively on the interactions within the target context).

Context	User 25-NN			Item 10-NN		
	Similarity	Coverage	Recall	Similarity	Coverage	Recall
reference		0.49156	0.33709		0.58758	0.34635
dow	identity	0.49258	0.29540	identity	0.77430	0.29856
dow	normalized	0.46220	0.35309	normalized	0.69414	0.36056
dow	cosine	0.44575	0.36404	cosine	0.60437	0.38478
daytime	identity	0.49941	0.29487	identity	0.72688	0.31814
daytime	normalized	0.46878	0.36372	normalized	0.64015	0.37225
daytime	cosine	0.45646	0.36814	cosine	0.60302	0.37941
hour	identity	0.49485	0.28119	identity	0.81218	0.25803
hour	cosine	0.41900	<b>0.38467</b>	cosine	0.55037	<b>0.39215</b>
hour	normalized	0.42010	0.38467	normalized	0.55552	0.39246
season	identity	0.49848	0.31193	identity	0.68402	0.32361
season	normalized	0.46093	0.36141	normalized	0.61399	0.37341
season	cosine	0.45368	0.36204	cosine	0.59821	0.37751
month	identity	0.49570	0.31140	identity	0.75911	0.28940
month	cosine	0.41951	<b>0.37878</b>	normalized	0.58876	0.38446
month	normalized	0.42103	0.37941	cosine	0.58016	<b>0.38509</b>

**Table 4.3.** Coverage and recall of context-aware versions of User 25-NN and Item 10-NN algorithms, Furniture dataset; the highlighted values correspond to the similarity matrices shown in Figure 4.11

The possible reason why even the nearly all-ones similarity matrices cause a significant recall increase is discussed later in Section 4.6.5, as this phenomenon occurs in other datasets too.

## 4.6.2 Posters Dataset

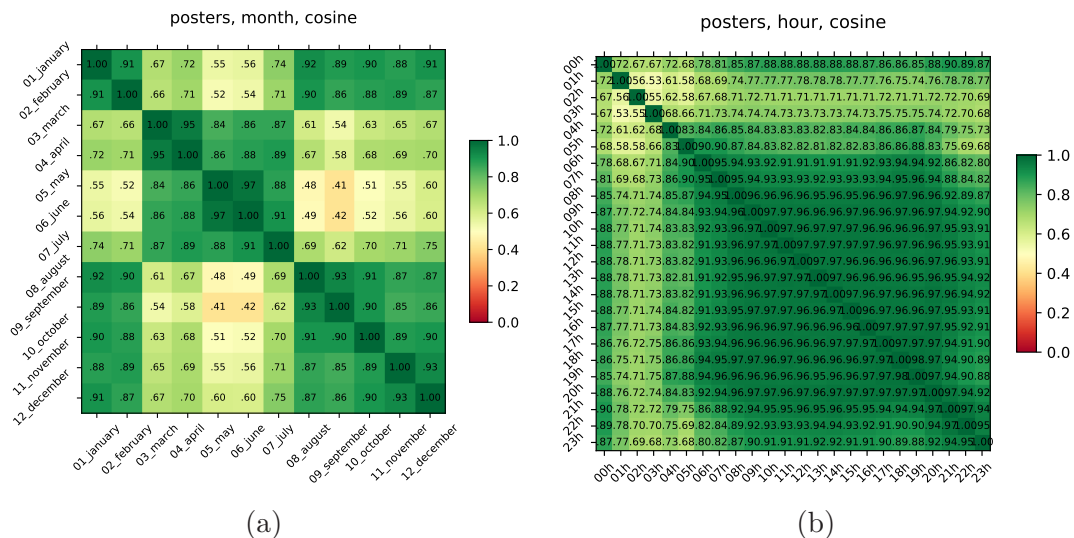
The monthly similarity matrix of the Posters dataset (Figure 4.12 a) shows even stricter division of warm (March to July) and cold (August to February) months. This is unexpected, because there is little reason for poster sales being different in cold and warm seasons of the year. However, the border is so sharp that the cause is probably not the weather but simply a regular change in the offer of goods taking place twice a year.

In Figure 4.12 b, it can also be seen that the popularity of items change during the night and in the early morning. Again, the cause may be having only a few interactions in these night periods, which in combination with Posters being a small dataset promotes the occurrence of more or less random items, causing the item-characteristics vectors being substantially different from the usual daytime item=characteristics.

The daytime and season similarity matrix resemble the appearance of the hourly and monthly matrices, respectively. The day of week matrix is close to all-ones matrix, having all values larger than 0.95.

Table 4.4 shows the measured coverage and recall for all studied contextual factors. The overall character of the results is similar to that of the Furniture dataset. Normalization to  $[0, 1]$  interval doesn't perform significantly better than a basic cosine-based similarity matrix, and algorithms using an identity matrix actually perform worse than the unmodified User and Item  $k$ -NN.

The hour and month context improve the recall the most, 23% in both cases. To a lesser extent, other contextual factors increase recall too, which will be explained later.



**Figure 4.12.** Similarity matrices of natural periodicities in Posters dataset computed using cosine similarity between item-characteristics of (a) month, (b) hour of a day

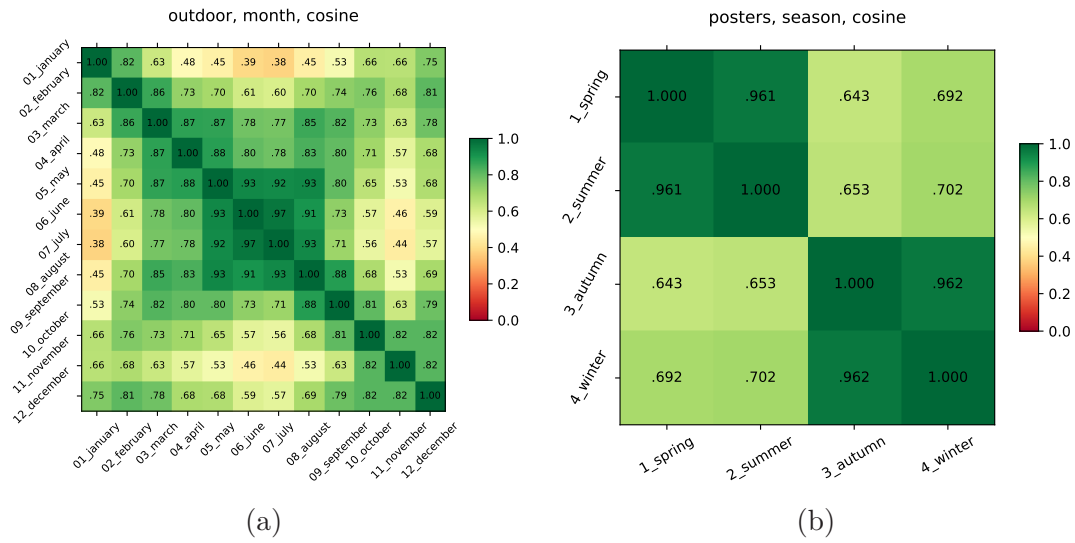
Context	Similarity	User 25-NN		Item 10-NN		
		Coverage	Recall	Similarity	Coverage	Recall
reference		0.46058	0.20239		0.64492	0.20074
dow	identity	0.51138	0.14977	identity	0.80788	0.14977
dow	normalized	0.43032	0.21049	normalized	0.70877	0.21638
dow	cosine	0.40450	0.21398	cosine	0.62715	0.22815
daytime	identity	0.50472	0.14315	identity	0.80316	0.16578
daytime	cosine	0.41477	0.21950	normalized	0.67990	0.22484
daytime	normalized	0.42449	0.21987	cosine	0.65547	0.22649
hour	identity	0.52443	0.14646	identity	0.79789	0.12309
hour	normalized	0.35786	0.24637	normalized	0.54386	0.23625
hour	cosine	0.35702	<b>0.24839</b>	cosine	0.53692	<b>0.23717</b>
season	identity	0.51610	0.18160	identity	0.78456	0.17498
season	normalized	0.45142	0.21877	normalized	0.73293	0.20313
season	cosine	0.42726	0.22208	cosine	0.68517	0.21674
month	identity	0.51749	0.18546	identity	0.80622	0.15492
month	cosine	0.36091	<b>0.24821</b>	normalized	0.64603	0.23091
month	normalized	0.37035	0.25225	cosine	0.60744	<b>0.23367</b>

**Table 4.4.** Coverage and recall of context-aware versions of User 25-NN and Item 10-NN algorithms, Posters dataset; the highlighted values correspond to the similarity matrices shown in Figure 4.12

### 4.6.3 Outdoor Dataset

Popularity of the items in the Outdoor dataset was expected to be highly dependent on the season of the year. This dependence is clearly projected into the month similarity matrix, shown in Figure 4.13 a. In contrary to the Furniture and Posters datasets, where there was a rather sharp border between Summer and Winter months, the transition is smooth in the Outdoor dataset. The smooth appearance agrees with the assumption that there are many season-specific items in the dataset. For instance, ski equipment





**Figure 4.13.** Similarity matrices of natural month and season periodicity in Outdoor dataset computed using cosine similarity between item-characteristics

is specific to Winter, but presumably the popularity of ski equipment already rises in Autumn.

The hour and daytime similarity matrix of the Outdoor dataset is similar to that of the Furniture dataset, shown in Figure 4.11 b, only the stripe of low similarities in night hours is much less distinct, the lowest value being 0.86. The matrix for day of week is close to an all-ones matrix.

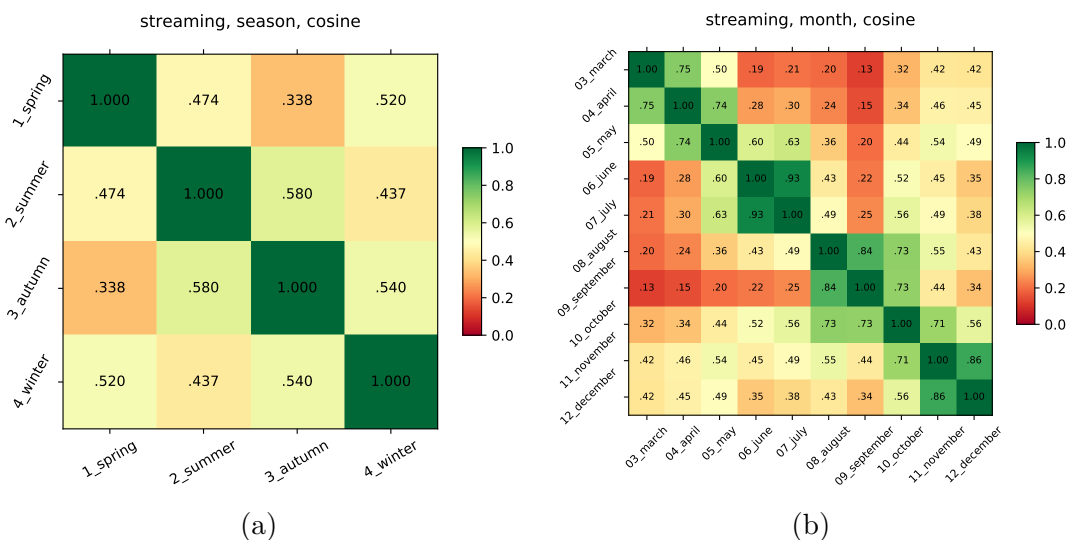
Context	User 25-NN			Item 10-NN		
	Similarity	Coverage	Recall	Similarity	Coverage	Recall
reference		0.71551	0.30261		0.77833	0.36402
dow	identity	0.72654	0.22014	identity	0.93874	0.33192
dow	normalized	0.66853	0.29643	normalized	0.84932	0.37293
dow	cosine	0.64517	0.30312	cosine	0.77521	0.37911
daytime	identity	0.72641	0.21112	identity	0.91084	0.34181
daytime	normalized	0.67891	0.30588	normalized	0.80143	0.37486
daytime	cosine	0.66385	0.32079	cosine	0.77210	0.37755
hour	identity	0.72706	0.22370	identity	0.95419	0.30213
hour	normalized	0.60091	0.38093	normalized	0.74069	0.38351
hour	cosine	0.59611	0.38180	cosine	0.73576	0.38373
season	identity	0.72187	0.29890	identity	0.83660	0.35639
season	normalized	0.68254	0.34973	normalized	0.78559	0.37380
season	cosine	0.66074	<b>0.35722</b>	cosine	0.75782	<b>0.37471</b>
month	identity	0.73108	0.31301	identity	0.86295	0.33842
month	cosine	0.61674	<b>0.37690</b>	cosine	0.73874	<b>0.37868</b>
month	normalized	0.62713	0.37820	normalized	0.75146	0.38031

**Table 4.5.** Coverage and recall of context-aware versions of User 25-NN and Item 10-NN algorithms, Outdoor dataset; the highlighted values correspond to the similarity matrices shown in Figure 4.13

The results of the experiments can be seen in Table 4.5, the recall achieved with the season and month matrices is highlighted. Although both season-aware and month-aware recommenders increase recall greatly (by 24% for User 25-NN and month similarity matrix), we can see that the highest recall was reached using an hour of day as a context, even when the hourly similarity matrix is very indistinctive. It turns out that the success of hourly similarity matrix is to a large extent a consequence of *bagging*, a well known technique for improving a model’s performance by splitting the training data into multiple parts. Bagging and its effect on experiments conducted in this chapter will be discussed in section 4.6.5.

#### 4.6.4 Streaming Dataset

The streaming dataset is very different from the other datasets. The first difference is visible from the season and month matrices, shown in Figure 4.14 (the month similarity matrix shows only months from March to December, because this dataset contains only interactions made during the span of these nine months).



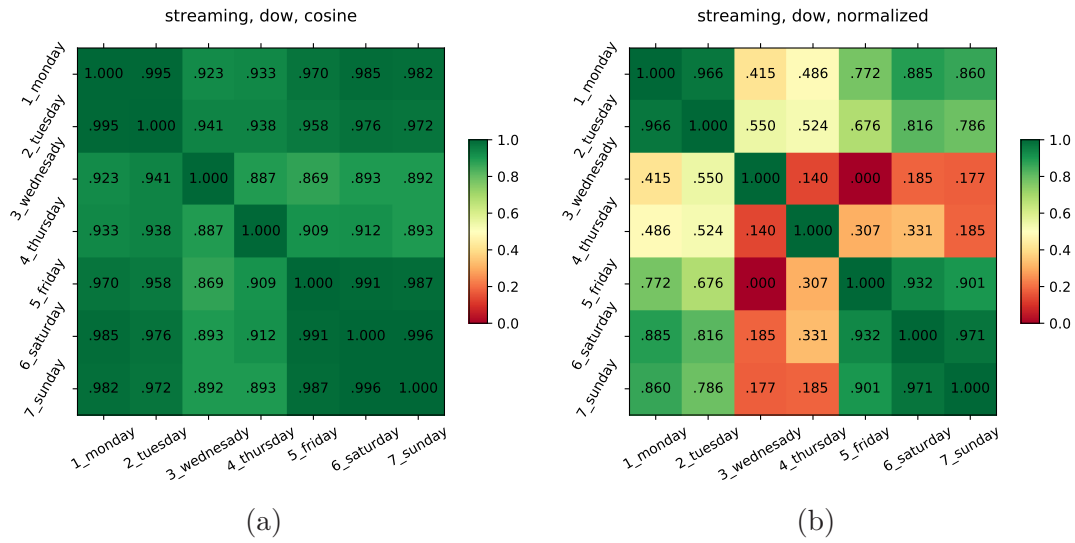
**Figure 4.14.** Similarity matrices of the Streaming dataset for (a) season, (b) month of a year

There is very little similarity in item characteristics of even two consecutive months. It means that the offerings of the products change in very quick pace and popularity of items older than several weeks declines to very low levels. This is also the reason why such good results were achieved in Section 4.5.

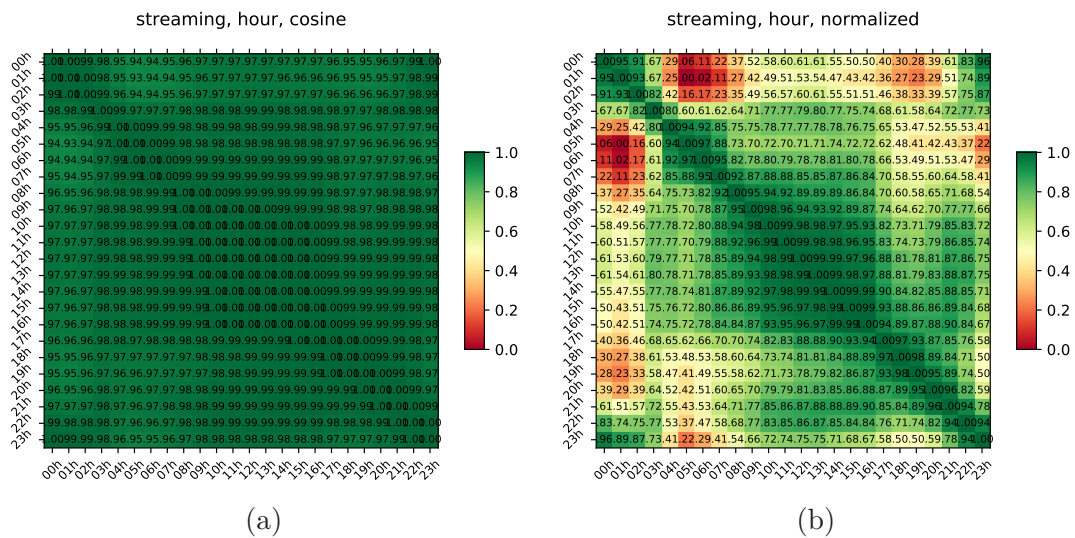
Figure 4.15 shows the day-of-week similarity matrix in its original and scaled version. Multimedia content consumed on Thursday and Wednesday differs from the content consumed in other days. It loosely resembles the cycle of weekend/workday, with Monday and Tuesday being unexpectedly similar to weekend.

The hour matrix is not as clear as in the case of Furniture or Posters datasets, but when scaling is applied to it, the shape is similar to that shown in Figure 4.12 b. An interesting feature of this matrix is the low-valued stripe at 5 o’clock, which is dissimilar to times from 19 pm until morning. It may be caused by cartoons present in this dataset, which aren’t viewed late in the evening.

The measured results are shown in Table 4.6. The best result (in terms of recall improvement), 14%, has been reached using an identity matrix and an Item 10-NN



**Figure 4.15.** Day of week similarity matrices of the Streaming dataset, (a) cosine-based, (b) scaled to  $[0, 1]$



**Figure 4.16.** Hour similarity matrices of the Streaming dataset, (a) cosine-based, (b) scaled to  $[0, 1]$

algorithm. The reason why identity matrices for season and month contextual factors keep up with the more sophisticated ones is the changeability of the dataset. The identity matrix actually causes a similar effect as temporal dynamics weighting, which has worked very well for Streaming dataset—the identity matrix causes the model to be trained only with the interactions made in the target month/season (in other words, with recent interactions). This can explain the good performance of month and season identity matrices, in contrast with their poor performance in other contextual factors or other, more stable datasets.

Scaling has had no substantial effect on recall, despite scaling of the hour and day-of-week similarity matrices uncovered interesting patterns in the data. It is unfortunate that the hour similarity matrix, which is very close to all-ones matrix, performed nearly

Context	User 25-NN			Item 10-NN		
	Similarity	Coverage	Recall	Similarity	Coverage	Recall
reference		0.34350	0.23838		0.38462	0.21953
daytime	identity	0.36036	0.18621	identity	0.37946	0.24016
daytime	normalized	0.34516	0.22863	normalized	0.33356	0.24489
daytime	cosine	0.34428	0.23205	cosine	0.33044	0.24529
dow	identity	0.37429	0.17217	identity	0.37420	0.23729
dow	normalized	0.35052	0.22412	cosine	0.34282	0.24212
dow	cosine	0.34331	0.22746	normalized	0.34662	0.24340
hour	identity	0.39690	0.13623	identity	0.40928	0.22394
hour	normalized	0.32255	0.22747	normalized	0.28269	0.23281
hour	cosine	0.32118	0.22772	cosine	0.28182	0.23347
month	identity	0.39476	0.22070	cosine	0.42701	0.23496
month	cosine	0.39797	0.25271	normalized	0.42633	0.23805
month	normalized	0.39846	0.25646	identity	0.45654	<b>0.25038</b>
season	identity	0.37420	0.22154	identity	0.42711	0.23980
season	cosine	0.37332	0.24660	normalized	0.41610	0.24940
season	normalized	0.37644	0.25049	cosine	0.41503	0.24991

**Table 4.6.** Coverage and recall of context-aware versions of User 25-NN and Item 10-NN algorithms, Streaming dataset; best recall improvement is highlighted

identically to its scaled variant, although these two matrices are very different. Again, this will be explained in the following section.

Another oddness of this dataset is that in many cases, the recall has been increased only when using Item  $k$ -NN as an underlying algorithm, not when User  $k$ -NN has been used. This may be connected to another distinctive property of this dataset, which is that users own many interactions. Approximately  $\frac{2}{3}$  of the users own more than 10 interactions each. This is unusual for the other studied datasets, in which only about 3% of the users own more than 10 interactions.

That means that in Streaming dataset, each context-specific User  $k$ -NN model has enough data to find a set of  $k$  most similar users to the target user, but the set can be very different between individual partial models. When scoring the items, each model votes for its own set of items, causing the recommendation to be less relevant. Item  $k$ -NN is more robust to this effect, because each item usually has many interactions, therefore the item-item similarity is more stable even when taking only a portion of the training data (belonging to the target context) into account.

The disbalance between User  $k$ -NN and Item  $k$ -NN didn't occur in other datasets, because users in other datasets usually have only a few interactions, so the various partial models often have nothing to recommend, causing only a few partial models to dominate the recommendation while others are scoring all items zero.

#### 4.6.5 Bagging

The experiments with segmentation did not come up to expectations. Although some of the similarity matrices look exactly as predicted and many of them display interesting properties of the datasets, the recall improvement does not reflect this. In some cases, the matrices which were close to all-one matrices reached comparable or even better

results than those which revealed some true periodicities in item popularity. The high recall measured for these matrices has probably been caused by *bagging* instead.

Bagging, or by full title *bootstrap aggregation*, is a meta-algorithm serving as an ensemble framework for basic algorithms such as User or Item  $k$ -NN. This method, first introduced in [20], aims to improve stability of a model, but it may also improve the accuracy metrics.

The idea of bagging is to generate multiple training subsets from the original training set using bootstraping. That's to say, each subset contains a random sample from the original set, with replacement. Multiple models are then trained, one on each training subset. Their outputs are combined by voting or averaging.

As can be seen, running the segmentation algorithm with a matrix close to all-ones matrix is a special case of bagging, where the training subsets do not overlap and each sample is used exactly once. This causes the success of segmentation even when using uniform similarity matrices.

To estimate the influence of bagging on the recall improvement, the measured results can be compared to models, of which similarity matrix is an all-ones matrix. This way, the weighting part of the algorithm is suppressed and only the influence of bagging remains in effect. The comparison of chosen best results with their counterparts with all-ones similarity matrix is shown in Table 4.7. Only the matrices which were expected to reflect the periodicity of a given dataset and which produced good results on their own are shown in the table.

Dataset	Context	User 25-NN recall		Item 10-NN recall	
		Cosine	All-ones	Cosine	All-ones
Furniture	hour	0.38467	<b>0.38551</b>	<b>0.39215</b>	0.39067
Furniture	month	<b>0.37878</b>	0.37772	<b>0.38509</b>	0.38151
Posters	hour	<b>0.24839</b>	0.24729	0.23717	<b>0.23735</b>
Posters	month	<b>0.24821</b>	0.24287	<b>0.23367</b>	0.22778
Outdoor	month	<b>0.37690</b>	0.37391	<b>0.37868</b>	0.37522
Outdoor	season	<b>0.35722</b>	0.35501	<b>0.37471</b>	0.37050
Streaming	dow	0.22747	<b>0.22754</b>	0.24212	<b>0.24282</b>

**Table 4.7.** Comparison of selected recall measurements with recall of all-ones matrices; the better of the two is highlighted

It can be deduced that although in most cases the models with cosine-based similarity matrices still produce better results than models with all-ones matrices, the difference between the two is far smaller than the difference between unmodified collaborative filtering algorithm and any of the context-aware modifications. That means most of the recall improvement can be attributed to bagging. Only about 1% recall increase is observed when all-ones matrices are replaced with cosine-based similarity matrices and the improvement isn't even guaranteed. Such improvement can't be considered reliable and worth implementing into contemporary recommender systems, as the comparable results can be achieved with bagging, which is a proven and in-depth researched method.

## Chapter 5

### Conclusion

The evaluation framework was successfully designed and implemented, along with User  $k$ -NN and Item  $k$ -NN collaborative filtering algorithms. Two classes of meta-algorithms with multiple variations and parameters were proposed and integrated into the framework. Recall and catalog coverage were measured for thousands of combinations on four real business datasets with implicit interactions. Important results were presented and discussed.

Both temporal dynamics and the segmentation techniques were evaluated with the time dimension of the data, although segmentation can work with any other contextual factor as well.

Experiments have shown that taking temporal dynamics into consideration increases recall of the recommender by 5–35%, depending on dataset. As expected, higher improvements were observed on highly dynamic datasets with frequent changes in item offering.

Exponential weighting function works the best, being closely followed by Gaussian weighting function. Linear weighting is too drastic and results in poor recommendation.


Segmentation methods with cosine-based similarity matrix improve the recommendation. Experiments with identity matrix ( $\mathbf{M} = \text{diag}(1, \dots, 1)$ ) resulted in decreased recall, which indicates that interactions made in different context than the target context are still valuable for the recommendation, even if not as much as the interactions made in the target context.

However, the success of the proposed segmentation method is possibly to a large extent caused by bagging, a well-known method of improving a model accuracy by splitting the training data into multiple sets, which is an inevitable side-effect of the proposed algorithm. Experiments with all-ones similarity matrix ( $\forall i, j \in C : \mathbf{M}_{i,j} = 1$ ) resulted in nearly as good results as the experiments with similarity matrices based on cosine similarities of item-characteristics of various contexts.

Nevertheless, the similarity matrices derived from cosine similarities between item-characteristics vectors turned out to be a great visualisation tool for discovering the structure of a dataset. In many cases, the predicted features of the datasets were clearly visible in the similarity matrices. It also helped to identify unexpected periodicities in some datasets.

The implemented framework is ready to be used in similar experiments. Its modular architecture allows for easy replacement of the base algorithms, opening room for experiments with, e.g., matrix factorization, and also development of wrapper algorithms similar to those used in this thesis.

The research has shown positive impact of incorporating context into collaborative filtering methods. Weighting the interactions with respect to their age improves the offline metrics of the recommendation. More specifically, the exponential weighting function with small values of  $\psi$  improved recall on all four studied datasets. Provided that there is very small computational overhead and the ease of wrapping an existing



algorithm into this context-aware extension, the proposed method is perfectly usable in business.

However, more research is needed for the segmentation methods. The measured similarity matrices show that some datasets are visibly periodic and some items are being used in a single context only, uncovering large potential for improvement of the recommendation. Nevertheless, the weighted ensemble method presented in this thesis seems to be insufficient to profit from the item popularity periodicity.

## References

- [1] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. New York: Springer Science+Business Media, LLC, 2011. ISBN 978-0-387-85819-7.
- [2] Shashikanth C C, and Parag Kulkarni. Region based Image Similarity using Fuzzy based SIFT Matching. *International Journal of Computer Applications*. 2013, 67 (3), 47-50.
- [3] Michael J. Pazzani, and Daniel Billsus. *Content-Based Recommendation Systems*. In: Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, eds. *The Adaptive Web*. Berlin, Heidelberg: Springer, 2007. ISBN 978-3-540-72078-2.
- [4] Shah Khusro, Zafar Ali, and Irfan Ullah. *Recommender Systems: Issues, Challenges, and Research Opportunities*. In: Mohamad Faizal Ab Jabal, Mustafa Man, and Mohd Rahim, eds. *A Comparative Study on Hough Transform Segmentation Approach for Outlier Iris Image*. Singapore: Springer, 2016. 1179-1189. ISBN 978-981-10-0556-5.
- [5] Tomáš Řehořek. *Manipulating Capacity of Recommendation Models in Recall-Coverage Optimization*. Ph.D. Thesis, Czech Technical University in Prague. 2018.
- [6] F. O. Isinkaye, Y. O. Folajimi, and B. A. Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*. 2015, 16 (3), 261–273. DOI 10.1016/j.eij.2015.06.005.
- [7] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender Systems Survey. *Knowledge-Based Systems*. 2013, 46 109–132. DOI 10.1016/j.knosys.2013.03.012.
- [8] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*. New York: Cambridge University Press, 2011. ISBN 978-0-521-49336-9.
- [9] Jon Herlocker, Joseph A. Konstan, and John Riedl. An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms. *Information Retrieval*. 2002, 5 (4), 287–310. DOI 10.1023/A:1020443909834.
- [10] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. *Item-based Collaborative Filtering Recommendation Algorithms*. In: *Proceedings of the 10th International Conference on World Wide Web*. New York, NY, USA: ACM, 2001. 285–295. ISBN 1-58113-348-0.
- [11] Mukund Deshpande, and George Karypis. Item-based top-N Recommendation Algorithms. *ACM Transactions on Information Systems*. 2004, 22 (1), 143–177. DOI 10.1145/963770.963776.
- [12] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. Incorporating Contextual Information in Recommender Systems Using



- a Multidimensional Approach. *ACM Transactions on Information Systems*. 2005, 23 (1), 103–145. DOI 10.1145/1055709.1055714.
- [13] Linas Baltrunas, and Francesco Ricci. *Context-based Splitting of Item Ratings in Collaborative Filtering*. In: *Proceedings of the Third ACM Conference on Recommender Systems*. New York, NY, USA: ACM, 2009. 245–248. ISBN 978-1-60558-435-5.
- [14] Linas Baltrunas, Bernd Ludwig, Stefan Peer, and Francesco Ricci. Context Relevance Assessment and Exploitation in Mobile Recommender Systems. *Personal Ubiquitous Computing*. 2012, 16 (5), 507–526. DOI 10.1007/s00779-011-0417-x.
- [15] Yehuda Koren. Collaborative Filtering with Temporal Dynamics. *Commun. ACM*. 2010, 53 (4), 89–97. DOI 10.1145/1721654.1721677.
- [16] B. Karahodža, D. Đonko, and H. Šupić. *Temporal dynamics of changes in group user’s preferences in recommender systems*. In: *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Opatija: IEEE, 2015. 1262–1266. ISBN 978-9-5323-3082-3.
- [17] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. *An Algorithmic Framework for Performing Collaborative Filtering*. In: *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: ACM, 1999. 230–237. ISBN 1-58113-096-1.
- [18] P. Cremonesi, R. Turrin, E. Lentini, and M. Matteucci. *An Evaluation Methodology for Collaborative Recommender Systems*. In: *2008 International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution*. Florence, Italy: IEEE, 2008. 224–231. ISBN 978-0-7695-3406-0.
- [19] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems*. 2004, 22 (1), 5–53. DOI 10.1145/963770.963772.
- [20] Leo Breiman. Bagging predictors. *Machine Learning*. 1996, 24 (2), 123–140. DOI 10.1007/BF00058655.



# Appendix A

## The Evaluation Framework and Measurements

The attached media contains three directories. One contains the source code and a PDF document with this thesis, second contains the source code of the evaluation framework developed for the demands of this thesis, and the third contains a file with all measured recall and catalog coverage values for various datasets and parameter combinations.

The file structure of the media is the following:

```
/
+- thesis/
| +- scheumar.pdf      The text of the thesis
| +- scheumar.tex     The main TeX source file of the thesis
| +- assignment.pdf   The assignment of the thesis
| +- chapters/
| | +- ...            The TeX files with the chapters of the thesis
| +- ...              Other dependencies of the CTU thesis template
+- framework/
| +- main.py          The main Python executable of the framework
| +- ...              Other Python source code files
+- measurements/
  +- measurements.db  A SQLite database with the measured results
  +- schema.txt       SQL schema of measurements.db
```

### A.1 The Evaluation Framework

The framework is implemented in Python 3 and it is dependent on the following external libraries: `numpy` (numeric computation), `scipy` (sparse matrices), `sqlite3` (access to the file with results), `matplotlib` (drawing the plots). The optional dependencies needed only if computing new results: `sklearn` (preprocessing algorithms), `psycopg2` (database access), `ruamel.yaml` (YAML parser for scripted plots).

The usage instructions can be invoked by passing the `--help` option:

```
python3 ./main.py --help
```

There are several subprograms runnable by the main program. The ones which only show the results (saved in `measurements.db`, which must be present in the same directory as the main program) don't need a source database. They are:

- `results` – Show recall and coverage measured for the specified attributes.
- `showsim` – Show the similarity matrix of a model with the specified attributes.
- `plotcr` – Show a recall/coverage plot for specified attributes. Multiple values can be specified for an attribute, they must be delimited by a colon.
- `plotscript` – Show a recall/coverage plot based on a preprogrammed pattern specified in `plotscripts.yaml`.

Example usage (see the built-in help for options):

```
./main.py result --s posters -m userknn -r 0.1 -k 100 -c month
./main.py showsim -s outdoor -c month -d cosine
./main.py plotcr -s furniture -c forget_exp -a user -p psi 1:3:7:14:365
./main.py plotscript -s p_i_f_u
```

The other subprograms need a Postgres database connection and work with the interaction data. The database must run on localhost, the connection can be configured by shell environmental variables `DB_USER` (user), `DB_PASSWORD` (password for the user), `DB_DATABASE` (database name). The table with the interactions must have `userid` (unique identifier of a user), `itemid` (ID of an item) and `timestamp` (time of the interaction) columns.

The subprograms working with the database are:

- **describe** – Show basic properties of the dataset (number of users, items, etc.).
- **compute** – Compute recall and coverage of the specified model. More attributes can be specified at once with values delimited by colons. The computation is executed for all possible combinations of the attributes.
- **ui** – A very basic user interface for visual evaluation of a model.

## A.2 Measurements

All measurements are saved into `measurements.db` file, which is an SQLite3 database file. It should not be needed to read this file directly, as the information can be accessed using the framework commands, but it can be practical to read the raw results when full power of the SQL language is needed.

The database file contains two tables: `measurements` with recall and coverage measurements and `msims` with similarity matrices. The columns of these tables are described in the attached file, `schema.txt`.