Assignment of master's thesis

Title:	Porovnání online a offline evaluačních metrik v doporučovacích systémech
Student:	Bc. Petr Kasalický
Supervisor:	Ing. Tomáš Řehořek, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

Explore the area of Recommender Systems with a focus on the online and offline evaluation of recommendation models.

Examine and analyze the application of matrix factorization for implicit feedback datasets [1] for recommendation using the k-NN algorithm.

Design an implement experimental framework for comparing the quality of offline metrics to the online metrics for a kNN algorithm using A/B testing.

Implement the proposed framework with all the necessary optimizations to make the framework capable of handling real-world, industrial datasets.

Perform a set of experiments on multiple different datasets and to compare the impact of different hyperparameterizations on both the offline (recall, popularity-stratified recall, catalog coverage) and online metrics (CTR).

Present, conclude the collected results and discuss their practical implications.

[1] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets.

Electronically approved by Ing. Karel Klouda, Ph.D. on 5 February 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Comparison of online and offline evaluation metrics in Recommender Systems

Bc. Petr Kasalický

Department of Applied Mathematics
Supervisor: Ing. Tomáš Řehořek, Ph.D.

May 6, 2021

Acknowledgements

First of all, I would like to thank my supervisor Ing. Tomáš Řehořek, Ph.D., for his valuable time and pieces of advice. I equally thank Recombee for the opportunity to use their system as part of the experiment and for renting computing servers, and Ing. Ladislav Martínek for help with the integration of the resulting framework. I thank Ing. Karel Klouda, Ph.D., for advices and proofreading of mathematical notation. A tremendous thank you also goes out to my family and girlfriend for their support during my studies.

Declaration

In Prague on May 6, 2021 . . . .

Citation of this thesis

Kasalický, Petr. *Comparison of online and offline evaluation metrics in Recommender Systems*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Cílem práce je prozkoumat doporučovací systémy a způsoby jejich vyhodnocení. Je kladen důraz na porovnání online a offline způsobů vyhodnocení, neboť jejich vztah je velmi sporný. Při výzkumu se běžně používá recall pro optimalizaci doporučovacích algoritmů. Avšak recall může trpět různými problémy a nemusí vždy odpovídat online metrikám, jako je míra prokliku. Toto tvrzení je experimentálně ověřeno za použití produkčního doporučovacího systému s cílem změřit korelaci mezi recallem a mírou prokliku. Jak ukazuje sada vyčerpávajících experimentů s velkým množstvím modelů a metrik na několika průmyslových datasetech, tak korelace mezi recallem a mírou proklikou není vždy zaručena. Vzniká tak velká otázka nad současnými metodami porovnávání modelů ve výzkumu. Jako částečné zlepšení offline metrik je představena nová metoda měření recallu, která lépe reflekujte sekvečnost interakcí stejně jako jejich nenáhodné rozdělení, a tím zvyšuje její podobnost s mírou prokliku.

Klíčová slova recall, míra prokliku, korelace, vyhodnocení, implicitní interakce, maticová faktorizace

Abstract

The goal of this work is to explore Recommender Systems and methods of evaluating them. The focus is on comparing online and offline approaches of evaluation, as their relationship is highly questionable. In research, recall is commonly used to optimize recommendation algorithms. However, recall can suffer from various problems and may not always correspond to online metrics such as click-through rate. This claim is experimentally verified by measuring the correlation between recall and the click-through rate using a production Recommender System. As shown by a set of exhaustive experiments with a large number of models and metrics on several industrial datasets, the correlation between recall and the click-through rate is not always guaranteed. This raises a big question about current methods of comparing models in research. As a partial improvement of offline metrics, a new approach of measuring recall is introduced to reflect better the sequence nature of interactions as well as their non-random distribution and increase correlation with the click-through rate.

Keywords recall, click-through rate, correlation, evaluation, implicit interaction, matrix factorisation

Contents

Introduction **1**

1 Analysis of Recommender Systems **3**

1.1 Interactions/feedback . . . 3

1.2 Recommendation algorithms . . . 5

1.2.1 Collaborative filtering . . . 6

1.2.2 Top-K recommendation . . . 8

1.2.3 Matrix factorisation . . . 9

1.2.4 Item-KNN . . . 13

1.3 Evaluation of RS . . . 13

1.3.1 Offline evaluation . . . 14

1.3.1.1 Evaluation data . . . 14

1.3.1.2 Rating prediction-based evaluation . . . 16

1.3.1.3 Ranking-based evaluation . . . 17

1.3.1.4 Other standard methods . . . 21

1.3.2 Online evaluation . . . 22

1.3.3 Comparison of online and offline evaluation metrics . . . 24

2 Proposed experiment **25**

2.1 Hyperparametrization of recall metric . . . 25

2.2 Experiment steps . . . 26

2.3 Description of experimental data . . . 29

3 Implementation **31**

3.1 Datasets . . . 31

3.2 Modules . . . 31

4 Results **37**

4.1 Hyperparameters . . . 37

4.2 Collected data . . . 38

4.3 Analysis of CTR 39
4.4 The effect of hyperparameters on recall 42
4.5 Recall vs CTR 49
4.6 Summary of results 55

5 Discussion **57**

Conclusion **59**

Bibliography **61**

A Acronyms **71**

B Contents of enclosed CD **73**

List of Figures

1.1 Example of a rating matrix with interaction vectors [1] 8
1.2 Illustration of MF for two matrices inspired by [2] 9

3.1 Overview of the experiment 36

4.1 Changes of CTR over time for dataset A 40
4.2 Changes of CTR over time for dataset B 40
4.3 Changes of CTR over time for dataset C 41
4.4 Changes of CTR over time for dataset D 41
4.5 Changes of CTR over time for dataset E 41
4.6 The development of recall over time for models 43
4.7 Comparison of β, validation method and recall for dataset A . . . 45
4.8 Comparison of β, validation method and recall for dataset B . . . 47
4.9 Comparison of β, validation method and recall for dataset C . . . 48
4.10 Comparison of β, validation method and recall for dataset D . . . 49
4.11 Comparison of β, validation method and recall for dataset E . . . 50

List of Tables

1.1 Overview of notation and terms defined for the RS 14
1.2 Confusion matrix for for general binary classification task 17

3.1 Description of datasets 32

4.1 Hyperparameters of ALS embedding for used models 38
4.2 Hyperparameters of ALS embedding for used models 39
4.3 Comparison of LOO and LLOO recall for $K = 5$ and $\beta = 1$ 44
4.4 Portion of models and parameters aggregated for each dataset where LOO > LLOO recall 44
4.5 Minimal, average and maximum correlation for selected parametrizations of recall 51
4.6 Comparison of correlation for each dataset 51
4.7 Minimal, average and maximum correlation for selected parametrizations of recall for datasets A, D, E 52
4.8 Comparison of correlations for different validation methods 53
4.9 Comparison of correlations for popularity penalisation 53
4.10 The relation between penalisation, validation and corelation across all datasets 54
4.11 The relation between penalisation, validation and corelation across datasets A, D, E 54

Introduction

Recommender Systems are very complex systems regarding the diversity of data processed. Advanced Recommender System is able to process the interactions of users, textual and binary attributes of items, or detailed user properties such as their current GPS location. Another complex task is to evaluate whether products identified as relevant are actually relevant. There are a number of metrics describing the quality of the recommendation algorithm. Most of them are taken from other areas, such as information retrieval or machine learning. Only a few really reflect the nature of the Recommender Systems from the perspective of practical application. Hundreds of researchers of recommendation algorithms optimise their models by a metric that does not always guarantee the practical use of the developed algorithm. This claim will be supported by a detailed analysis from a theoretical perspective. An experiment will then be conducted to confirm or refute this hypothesis.

First, this thesis will describe the Recommender Systems, with an emphasis on collected interactions and their problems. The category of recommendation algorithms based on interactions and matrix factorisation will be described in detail. Subsequently, the methods of evaluating Recommender Systems will be presented in a detailed survey. The benefits and problems of each evaluation method will be highlighted. Chapter 2 will introduce new methods of evaluation. The proposed methods will be experimentally compared, and it will be verified whether they are beneficial in optimising the recommendation algorithms in practical applications. A framework capable of processing large amounts of data will be implemented for the experiment. The structure of the framework will be presented in Chapter 3. Finally, Chapter 4 will present the results of the experiment and the conclusions on the properties of the different methods of evaluation.

Chapter 1

Analysis of Recommender Systems

The chapter will begin with a brief summary of the history and evolution of the Recommender Systems (RS). Subsequently, the different types of user interactions with the Recommender System will be detailed in Section 1.1. In Section 1.2, the basic approaches and principles of recommendation algorithms such as collaborative filtering, Top-K recommendation, and matrix factorisation will be presented. These basic principles will be referred to in Section 1.3 for a detailed explanation of the different ways how to evaluate the RS.

According to [3], RS originate in the Usenet system, which was created in 1980 to manage shared discussion groups. In the Usenet system, this first RS was tasked with filtering users' posts to highlight important ones. Today's modern RS handle far more use cases, both in system and user terms. They are used as a tool for collecting ratings, as a trusted authority for selecting goods (for example, concerning spare parts' compatibility) or connecting multiple users with the same interest [4, 5].

The scope of RS is truly enormous these days. As early as 1999, article [6] described how large e-commerce platforms such as Amazon, eBay, or Levis use RS to improve customer navigation between products. Since 2006, RS has also started to appear in the field of online newspapers [7] and is now also used in applications with user-generated content, where is no unification of text, images, and other features [8].

1.1 Interactions/feedback

Feedback from users is an integral part of the Recommender System. The feedback, also called interaction, produced by a user for certain content (products, songs, newspaper articles) serves as input for a whole category of recommendation algorithms called collaborative filtering. There are two distinguished types of feedback: implicit feedback (IF) and explicit feedback (EF) [4, p. 293].

The actions the user natively performs when working with the system are considered as implicit interactions. Types of IF vary from system to system, domain to domain, and develop over time. For example, according to [9], the following actions are common IF for the e-commerce domain: *purchase*, *cart addition*, *bookmarks*, *detail view.* For a platform to read and buy electronical books, publication [10] measures metrics such as *reading time of a content* or *duration of the session/content size.* For streaming music, article [11] counts, among other things, the number of times a song is played by a given user. The large-scale experiment was conducted in [12], where a custom browser called *The Curious Browser* was created, and the following feedback was collected from test users: *time spent on a page*, *time spent moving the mouse*, *the number of mouse clicks* and *time spent scrolling.* The movement of the mouse as the IF is also mentioned in [13]. Although it may seem from the listed cases that implicit feedback is mainly collected in modern applications, it is not true. The implicit feedback was also used in the Usenet mentioned above [14]. Likewise, the 2000 work [15] describes an Internet user's analysis by parsing website logs. While this diversity of interactions for individual systems provides an advantage in better understanding user actions, it is less theoretically supported [11]. Moreover, there is no standard in the representation of the IF. An interaction can be represented by a unary value, binary value, integer, decimal, or a sequence of coordinates in case of mouse movement. IFs by their definition are only positive or missing, never negative [16, 17]. Another drawback of implicit interactions is their not-uniform distribution. Typically, popular products/songs/articles or products promoted by RS will have far more implicit interactions because far more people have interacted with them. This phenomenon is called a missing-not-at-random (MNAR) problem and causes bias in recommendation algorithms. It needs to be taken into account in the evaluation [18, 19]. The last major obstacle to be considered in the experiments described in this thesis is a large amount of noise described in [20].

The second type of interaction is explicit feedback represented by a numeric value. An example is a product rating using 1 - 5 stars. The paper [17] states that explicit feedback expresses preferences, while implicit feedback shows confidence. According to the article [21], the Likert scale is often used to gain explicit feedback. It was introduced in 1932 in the publication [22] as a five-value scale and is used to analyse preferences even in today's sociological surveys. Contrary to implicit interaction, explicit feedback has the advantage that it can be both positive and negative (for example, rating a movie with a star or two). It is also estimated that explicit feedback is generally more accurate than implicit feedback to understand a user [23]. However, this is not true in all cases. The paper [24] claims that EFs are less accurate than IFs in some cases due to the fact that there are usually fewer of them and they do not typically evolve over time. It also claims that while collecting EFs using questionnaires, there is a psychological effect to be considered that people *“describe themselves more as what they would like to be than what they actually are”*. Similarly, EFs are inappropriate for reciprocal recommending, that is, when users are recommended to other users, such as an online dating site [25]. The (almost) consistent methodology for processing and evaluating EFs makes them more suitable for academic research. In addition, most of the commonly used benchmarks and datasets (e.g., MovieLens, Netflix Price) for evaluating recommendation algorithms contain only explicit ratings.

For further explanations, it is necessary to formally denote interactions. The set of interactions (both explicit and implicit) between users and items is denoted as

$$F = \{f_1, \dots, f_p\}. \tag{1.1}$$

A single interaction is denoted as

$$f_j \in (U \times I \times \mathbb{Z}_t \times \mathbb{R}_v) \text{ for } \forall j \in \{1, \dots, p\}, \tag{1.2}$$

where:

- $U = \{u_1, \dots, u_m\}$ is a set of users of the RS,
- $I = \{i_1, \dots, i_n\}$ is a set of items which can be recommended,
- $\mathbb{Z}_t$ is the set of integers expressing the timestamp when the interaction was performed,
- $\mathbb{R}_v$ is the set of real numbers expressing the numerical value of the interaction,
- p is a total number of interactions.

For a more practical approach to the individual components of an interaction, the interaction can be also denoted as a tuple

$$f_j = (u_j, i_j, t_j, v_j). \tag{1.3}$$

As has been said, some implicit interactions should be represented by a unary or binary value, but in this work, they will be converted into real numbers for simplification purposes. A detailed list of the interactions distinguished by this work, including their assigned numerical values, will be provided in Section 3.

1.2 Recommendation algorithms

The recommendation algorithms (RA) used in the practical part of this work will be presented in this section. Specifically, an approach called collaborative filtering will be described in detail first. Then it will be shown as this approach can be combined with a task called Top-K recommendation. The end of this section will be devoted to a technique called matrix factorisation.

1.2.1 Collaborative filtering

The introduction of Section 1.1 mentioned a group of RAs based on interactions called collaborative filtering (CF). It is not the only approach. There are others, such as content-based recommendations, where item and user attributes are used to calculate recommendations, but they are not relevant to this work. However, CF is important to mention, as it is used in the practical experiment of the thesis.

According to [26], the term collaborative filtering was first used in the Tapestry system described in the 1992 article [27]. It is also sometimes called *people-to-people correlation*, which better describes its principle [16, p. 12]. The basic idea behind CF is that the behaviour of an individual user is not unique. There are groups of users that are interested in the same content. Thus, if a new user comes in and RS recognised that the user belongs to the particular group, RS will recommend items that other group members also liked. This approach is called user-based collaborative filtering. An alternative is item-based CF, where RS finds items similar to those that a user liked in the past and recommend them [28, p. 38].

Both approaches are based on a rating matrix, which is also sometimes called an interaction matrix or user-item matrix [29]. In this work, it will be denoted as

$$R \in \mathbb{R}_?^{|U| \times |I|}, \tag{1.4}$$

where $\mathbb{R}_? = \mathbb{R} \cup \{?\}$ is an artificially extended set of real numbers by a value ? representing an unknown (unobserved) value. Although the indexes of the matrix should be integers in the range specificed by the size of the matrix, the elements of the sets U and I are used as indexes for the matrix R. Namely,

$$r_{u,i} \in \mathbb{R}_?$$

denotes the rating value for the user $u \in U$ and the item $i \in I$. Expression

$$r_{u,:} = (r_{u,i_1}, \ldots, r_{u,i_n})$$

represents a sequence of rating values of the user $u \in U$ and similarly

$$r_{:,i} = (r_{u_1,i}, \ldots, r_{u_m,i})^T$$

denotes a sequence of rating values of the item $i \in I$.

As defined at the end of Section 1.1, the RS can contain any number of interactions for each unique user-item pair. However, the rating matrix R can contain only one value for a user-item pair. If the RS recognises only one type of explicit feedback, its value is typically also the matrix's value. In the case of a unary IF (such as the *detail view*), value $r_{u,i}$ could be the number of such interactions for the user u and the item i [30]. When RS distinguishes more types of interactions, it is common practice to take the set of all interactions between the user u and the item i

$$F_{u,i} = \{(u_j, i_j, t_j, v_j) \mid (u_j, i_j, t_j, v_j) \in F : u_j = u \wedge i_j = i\}$$

and aggregate them into one value

$$r_{u,i} = \zeta(F_{u,i}),$$

where ζ is an aggregation function

$$\zeta : \{0,1\}^{|U|\times|I|\times\mathbb{Z}_t\times\mathbb{R}_v} \to \mathbb{R}_?. \tag{1.5}$$

A simple example of function ζ is a sum of the values:

$$r_{u,i} = \begin{cases} \sum\limits_{(u_j,i_j,t_j,v_j)\in F_{u,i}} v_j & \text{if } |F_{u,i}| > 0, \\ ? & \text{otherwise.} \end{cases} \tag{1.6}$$

Once the rating matrix was presented, it is possible to describe the rating prediction (RP) task. Namely, for the RP defined as

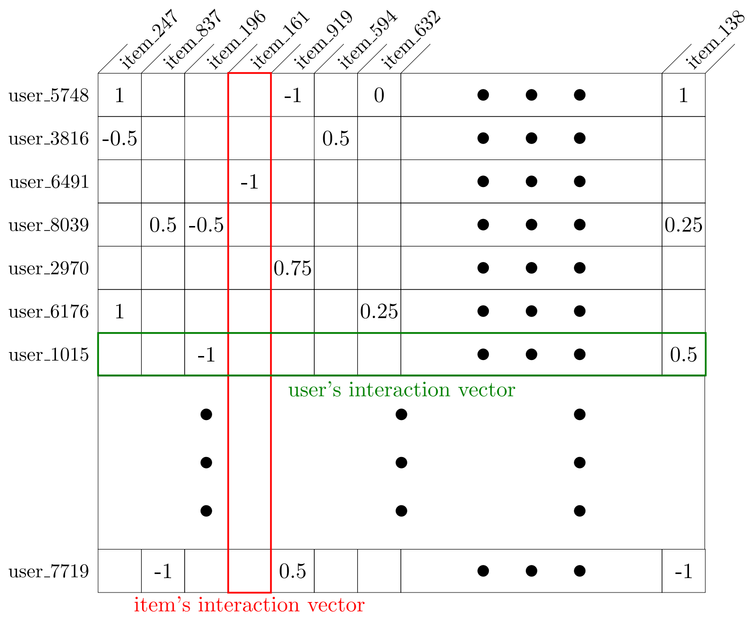
$$\tau : \mathbb{R}_?^{|U|\times|I|} \to \mathbb{R}^{|U|\times|I|}, \tag{1.7}$$

the goal is to replace the missing (in the case of IF) or unknown (in the case of EF) value ? in the rating matrix R by a value $\hat{r}_{u,i}$ while trying to keep $\hat{r}_{u,i} \approx r_{u,i}$ for $r_{u,i} \neq ?$.

The rating prediction task is a stand-alone recommendation task but also an optional step for item-based CF, which will be now presented in more detail. As previously described, the item-based CF approach takes a set of items that were interacted by the user u according to the R rating matrix and finds similar items to them. The most common method to measure the similarity of items is to measure the similarity of vectors representing items. An example of a vector representing an item is the vector $r_{:,i}$ that is also illustrated in Figure 1.1 and called *item's interaction vector*. It should be recalled that the $r_{:,i}$ vector is taken directly from the R rating matrix and is therefore typically very sparse. A very similar example of this is vector $\tau(R)_{:,i} = (\tau(r_{u_1,i}), \ldots, \tau(r_{u_m,i}))$, which is dense due to the fact that missing values were predicted. More examples of how to represent an item based on its interactions will be given in Section 1.2.3.

There are several commonly used ways to measure the similarity of vectors. Popular metric for very sparse vectors like $r_{:,i}$ is a cosine similarity [31, 32] calculated according to [33] as

$$cos(i_1, i_2) = \frac{\sum\limits_{\substack{u\in U \\ r_{u,i_1}\neq ? \\ r_{u,i_2}\neq ?}} r_{u,i_1}\, r_{u,i_2}}{\sqrt{\sum\limits_{\substack{u\in U \\ r_{u,i_1}\neq ?}} r_{u,i_1}^2}\sqrt{\sum\limits_{\substack{u\in U \\ r_{u,i_2}\neq ?}} r_{u,i_2}^2}}. \tag{1.8}$$



	item_247	item_837	item_196	item_161	item_919	item_594	item_632		item_138
user_5748	1				-1		0	● ● ●	1
user_3816	-0.5					0.5		● ● ●	
user_6491				-1				● ● ●	
user_8039		0.5	-0.5					● ● ●	0.25
user_2970					0.75			● ● ●	
user_6176	1						0.25	● ● ●	
user_1015			-1					● ● ●	0.5
			⋮				⋮	⋮	
user_7719		-1			0.5			● ● ●	-1

Figure 1.1: Example of a rating matrix with interaction vectors [1]

Following the MNAR problem (see Section 1.1), the article [31] describes that a similarity metric should be normalised so that popular items are not favoured. Cosine similarity does that, which is also one of the reasons why it is commonly used.

1.2.2 Top-K recommendation

A Top-K recommendation is an algorithm, where recommendable items are scored, but only K items with the highest score are recommended to the user [34]. It is a prevalent task in industrial applications of RS. An example is an e-shop website where each product has a detailed page. Below the product description there is an area with K places where recommended similar products are displayed. Score for each item does not have to be measured only by the cosine similarity of the interaction vectors. For example, the article [35] represents a Top-K recommendation based on implicit interactions using graphs.

However, this work will focus on item-based CF as it is used in the practical part of the thesis. Item-based CF has several advantages over user-based CF, such as better explainability [17] or scalability, but it is less accurate [31]. Moreover, according to the article [36], it suffers from long-tail effect described in [37].

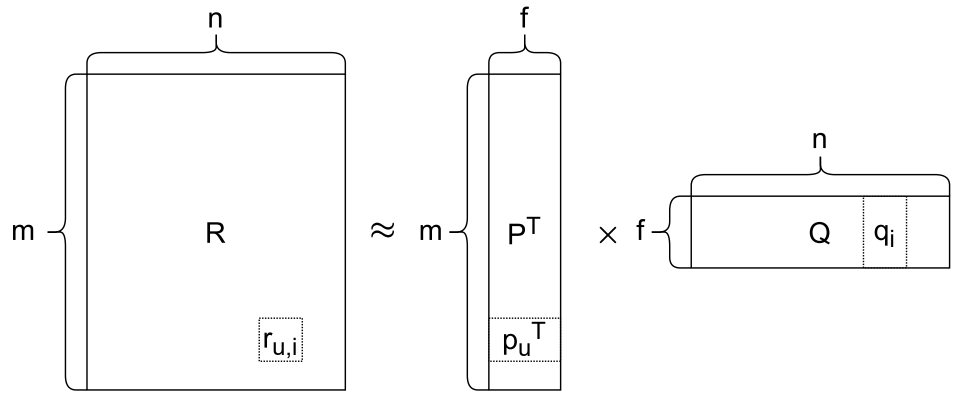


Figure 1.2: Illustration of MF for two matrices inspired by [2]

1.2.3 Matrix factorisation

In addition to item-based and user-based approaches of CF, there are other two commonly recognised approaches: neighbourhood methods and latent factor models (LFMs). The latter is built on the assumption that there is a *"low-dimensional representation of users and items where user-item affinity can be modelled accurately"* [38]. A problem when working with a rating matrix is its size, sparsity and noise caused by the noise of interaction data (see Section 1.1). LFM partially solves these problems by attempting to explain the values in the rating matrix *"by characterising both items and users on, say, 20 to 100 factors inferred from the rating patterns"*. In some cases, according to [26], found low-dimensional factors may contain explainable features such as the film genre. The article [39] argues that latent vectors obtained by LFM are extremely hard to interpret. The most popular LFM model according to [39] is matrix factorisation (MF), which has gained popularity thanks to the Netflix Prize contest [40]. As the name suggests, MF is based on the idea that the rating matrix R is factorised into two or more matrices. According to [41], the main representative of MF techniques for more than two matrices is the Singular Value Decomposition (SVD) method presented in [42], but since it will not be used in this thesis, it will not be described in detail. In the case of two matrices, the predicted rating for the item-user pair is calculated as $\hat{r}_{u,i} = f(p_u, g_i)$, where

- $p_u \in \mathbb{R}^f$ is a low-dimensional latent vector for user u,
- $q_i \in \mathbb{R}^f$ is a low-dimensional latent vector for item i,
- f can be a dot product but also some nonlinear function.

The basic idea of decomposing the matrix R into two smaller matrices denoted as $P \in \mathbb{R}^{f \times m}$ and $Q \in \mathbb{R}^{f \times n}$ is also shown in Figure 1.2. The book

[43, p. 152] points out that even a MF with a linear function can be represented by a neural network. This architecture can be generalised even further into the approach called Generalized Matrix factorisation (GMF), in which a multilayer perceptron is used instead of the dot product [44].

Until here, it has been mentioned how to predict the rating $\hat{r}_{u,i}$ value once the latent vectors p_u and q_i are known. Now it will be shown how to get those latent vectors. For now it is assumed that the function f is the dot product, $f(p_u, g_i) = p_u^T g_i$. The goal is to minimise the difference between the predicted rating $\hat{r}_{u,i}$ and the actual rating $r_{u,i}$ for known values $r_{u,i} \neq ?$ of the rating matrix. In the case of matrix factorisation, the difference between the predicted value and the actual value of the rating is commonly measured using a Mean Squared Error (MSE):

$$E_{u,i}^{MSE} = (r_{u,i} - \hat{r}_{u,i})^2 = (r_{u,i} - p_u^T g_i)^2.$$

The article [17] mentions that some papers use modified MSE to prevent overfitting:

$$e_{u,i}^{MSE-\lambda} = (r_{u,i} - p_u^T g_i)^2 + \lambda(\|p_u\|^2 + \|g_i\|^2),$$

where λ is a hyperparameter setting the degree of regularization. For matrices $P = (p_1, \dots, p_m)$ and $Q = (q_1, \dots, q_n)$ the loss function is then

$$L(P, Q) = \sum_{\substack{u \in U \\ i \in I \\ r_{u,i} \neq ?}} [(r_{u,i} - p_u^T g_i)^2 + \lambda(\|p_u\|^2 + \|g_i\|^2)]$$

and the optimisation task is

$$P, Q = \operatorname*{argmin}_{\substack{P \in \mathbb{R}^{f \times m} \\ Q \in \mathbb{R}^{f \times n}}} L(P, Q). \tag{1.9}$$

The two most popular methods to solve (1.9) are Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS). Only the latter is relevant to the practical experiments described in this work, and so SGD will not be explained in more detail.

ALS is based on a simple algorithm that the first matrix P is fixed and the second matrix Q is optimised. Then Q is fixed and P is optimised. These two steps are repeated. For the first step, when the Q matrix is optimised, the following applies:

$$\frac{\partial L}{\partial q_i} = -2 \sum_{\substack{u \in U \\ r_{u,i} \neq ?}} (r_{u,i} - p_u^T q_i) p_u + 2\lambda q_i. \tag{1.10}$$

For vector $r_{:,i}^{\neq ?}$ for the item i defined as:

$$r_{:,i}^{\neq ?} = (r_{u_1,i}^{\neq ?}, r_{u_2,i}^{\neq ?}, \dots, r_{u_m,i}^{\neq ?})^T \in \mathbb{R}^{m \times 1},$$

$$r_{u,i}^{\neq ?} = \begin{cases} r_{u,i} & r_{u,i} \neq ?, \\ 0 & r_{u,i} = ?, \end{cases}$$

can be (1.10) calculated as:

$$\frac{\partial L}{\partial q_i} = -2P(r_{:,i}^{\neq ?} - P^T q_i) + 2\lambda q_i. \tag{1.11}$$

Since the matrix P is constant in the first step of the ALS algorithm described by (1.11), q_i vector that minimises the function L can be found in the closed form:

$$\begin{aligned} -P(r_{:,i}^{\neq ?} - P^T q_i) + \lambda q_i &= 0 \\ -Pr_{:,i}^{\neq ?} + PP^T q_i + \lambda q_i &= 0 \\ PP^T q_i + \lambda q_i &= Pr_{:,i}^{\neq ?} \\ (PP^T + \lambda I) q_i &= Pr_{:,i}^{\neq ?} \\ q_i &= (PP^T + \lambda I)^{-1} Pr_{:,i}^{\neq ?}, \end{aligned}$$

where I is an identity matrix with dimensions $f \times f$. Similarly as the first step, the second step of ALS consists of the optimisation of matrix P while matrix Q is fixed:

$$r_{u,:}^{\neq ?} = (r_{u,i_1}^{\neq ?}, r_{u,i_2}^{\neq ?}, \ldots, r_{u,i_n}^{\neq ?}) \in \mathbb{R}^{1 \times n}$$

$$\frac{\partial L}{\partial p_u} = -2 \sum_{\substack{i \in I \\ r_{u,i} \neq ?}} (r_{u,i} - p_u^T q_i) q_i + 2\lambda p_u$$

$$\frac{\partial L}{\partial p_u} = -2Q(r_{u,:}^{\neq ?T} - Q^T p_u) + 2\lambda p_u \tag{1.12}$$

$$\begin{aligned} -Q(r_{u,:}^{\neq ?T} - Q^T p_u) + \lambda p_u &= 0 \\ -Qr_{u,:}^{\neq ?T} + QQ^T p_u + \lambda p_u &= 0 \\ QQ^T p_u + \lambda p_u &= Qr_{u,:}^{\neq ?T} \\ (QQ^T + \lambda I) p_u &= Qr_{u,:}^{\neq ?T} \\ p_u &= (QQ^T + \lambda I)^{-1} Qr_{u,:}^{\neq ?T}. \end{aligned} \tag{1.13}$$

First and second steps are repeated until the convergence criterion is met:

$$q_i \leftarrow (PP^T + \lambda I)^{-1} Pr_{:,i}^{\neq ?}, \tag{1.14}$$

$$p_u \leftarrow (QQ^T + \lambda I)^{-1} Qr_{u,:}^{\neq ?T}. \tag{1.15}$$

The ALS algorithm has been described for factorisation of a rating matrix composed of explicit interactions. As mentioned in Section 1.1, properties of implicit interactions differ from explicit ones in many things. One important is that IFs are never negative, just positive, or missing. Article [17] takes this into account and describes how to calculate matrix factorisation for a rating matrix containing implicit interactions. Specifically, the article proposed a method for checking and setting of confidence in the rating matrix. For that purpose, the preference matrix was created to indicate the positions of ratings in the rating matrix. Using already defined variables in this work with the notation inspirated by Thesis [2], preference matrix $Z \in \{0,1\}^{|U|\times|I|}$ can be defined by the formula:

$$z_{u,i} = sgn(\mid F_{u,i} \mid). \tag{1.16}$$

If user u interacted at least once with item i, then $z_{u,i}$ is equal to 1, otherwise it is 0. The confidence matrix $C \in \mathbb{R}^{|U|\times|I|}$ is defined in a similar way:

$$c_{u,i} = 1 + \alpha r_{u,i}^{\neq ?}, \tag{1.17}$$

where $\alpha \in R$ is a hyperparameter determining the confidence of $z_{u,i}$ observation. Subsequently, the matrix factorisation can be calculated using the ALS method, but with some adjustments. Specifically, the loss function is defined as:

$$L_{imp}(P,Q) = \sum_{\substack{u\in U\\ i\in I}} c_{u,i}(z_{u,i} - p_u^T g_i)^2 + \lambda(\sum_{u\in U} \|p_u\|^2 + \sum_{i\in I} \|g_i\|^2) \tag{1.18}$$

and the optimisation task to find matrices of latent vectors for users and items is:

$$P,Q = \operatorname*{argmin}_{\substack{P\in\mathbb{R}^{m\times f}\\ Q\in\mathbb{R}^{f\times n}}} L_{imp}(P,Q). \tag{1.19}$$

Similar to (1.10) and (1.12), one matrix can be fixed and the other minimised. In this case, publication [17] shows that the update steps are:

$$p_u = (QC^uQ^T + \lambda I)^{-1}QC^u z_u^T, \tag{1.20}$$

$$q_i = (PC^iP^T + \lambda I)^{-1}PC^i z_i^T, \tag{1.21}$$

where $C^u \in \mathbb{R}^{|I|\times|I|}$ is a diagonal confidence matrix for users with values $C^u_{i,i} = c_{u,i}$ and similarly $C^i \in \mathbb{R}^{|U|\times|U|}$ is a diagonal confidence matrix for items with values $C^i_{u,u} = c_{u,i}$. (1.20) contains a calculation of QC^uQ^T which is usually expensive to compute. To accelerate the computation, an improvement is proposed:

$$QC^uQ^T = QQ^T + Q^T(C^u - 1)Q,$$

because QQ^T does not depend on u and thus can be precomputed. Furthermore computation of $Q^T(C^u - I)Q$ is cheaper since $C^u - I$ has only $\mid \{z_{u,i} \mid u \in U, i \in I : z_{u,i} > 0\} \mid$ non-zero values. The similar optimisation of calculation can be used for PC^iP^T from (1.21). These optimisations are implemented in the Python package *Implicit*, which was used in the experimental part of this thesis to calculate ALS embeddings for items.

1.2.4 Item-KNN

Embeddings of items calculated using ALS can be used for Top-K recommendation algorithms. (1.8) presented how to calculate the cosine similarity of two items using their interaction vectors from the rating matrix. However, the similarity of items can also be measured as the similarity of ALS embeddings. The measured similarities can be then used as a distance metric for K nearest neighbours (NN) algorithm. An instance of the Top-K recommendation algorithm called item-KNN works as follows:

- A set of items I' interacted by the user u is collected.
- For each item $i \in I'$, similarities to all other items are measured. The calculated similarities are summed up across all items in set I'.
- K items with the highest overall similarity are selected for a Top-K recommendation. Selected items are declared relevant, and the remaining items are declared not relevant.

1.3 Evaluation of RS

This section will first explain the motivation behind the importance of examining the evaluation of RS. Subsequently, the two approaches of evaluation (offline, online) will be listed and described in detail in separate subsections. At the end of the section, the two approaches will be compared.

The RS consists of input data (interactions), recommendation algorithms, and evaluation metrics that evaluate the quality of the algorithms created. It is not possible to systematically create an excellent RA when the evaluation metric is wrong. There is no universal RA that works well on all different datasets and users. Every time a RA is created, its quality has to be measured. There are many RAs, even infinitely many, when the hyperparameters of individual algorithms are included. Evaluation metrics can be used to select the best algorithm. If the metric is more accurate (it will be specified later what it means), then the RAs are more successful in real applications, often resulting in increased profits of the platform running the RS [45]. Ideal evaluation metrics should evaluate the entire system and take into account the platform's high-level goals such as the number of clicks on items, the number of items purchased, the number of advertisements viewed, customer lifetime value, etc. [33]. However, these metrics are really hard to define, they are domain-dependent, and their results are unreproducible. Therefore, they are not suitable for research.

There are two commonly recognised approaches for the evaluation of RS. Namely, they are: offline evaluation of static data and corresponding evaluation metrics, and online evaluation using a live system and monitoring the implicit interactions of a real user or asking them to use popup surveys for explicit ratings of the recommended items [46]. There are alternatives where real users are not needed to evaluate a live system. For example, article [47] presents a framework in which recommendations are generated and evaluated using reinforcement learning (RL) based on historical data. In an oversimplified interpretation, RL creates an artificial user that can generate feedback for a recommendation engine instead of a real user.

1.3.1 Offline evaluation

The first approach to evaluate a recommendation algorithm is called offline evaluation. It is based on the idea that the evaluation metric uses the already collected and fixed set of interactions F defined in Section 1.1. The work [41] describes two possible situations: rating prediction-based evaluation (RPBE) and ranking-based evaluation (RBE). However, it is necessary to first describe what data must be used to evaluate the RA.

1.3.1.1 Evaluation data

Care must be taken to ensure the correct selection of the data used in the evaluation of the Recommender System. The methodology for evaluating RA using offline methods is the same as for other models trained with supervised learning. The basic rule is that the evaluation of RA should never be done on the data (interactions) that were used to create the RA. With that in mind, the interactions need to be divided into a training set and a test set.

Due to a large number of notations and defined terms in the previous sections, which now need to be adjusted, there is Table 1.1 repeating their meaning.

Notation	Description	Defined by
F	set of all interactions available in the RS	(1.1)
I	set of items which can be recommended	(1.2)
U	set of users of the RS	(1.2)
R	rating matrix	(1.4)
ζ	function aggregating a subset of interaction to one number	(1.5)

Table 1.1: Overview of notation and terms defined for the RS

Until now, the recommendation algorithms have been described in Section 1.2, and defined symbols were repeated along with its description in Table 1.1. For simplicity, algorithms were trained (for example, by the ALS method) on a set of all interactions. Similarly, (1.8) describes how to measure the similarity of two vectors that contain all available interactions. However, if the goal is also to evaluate an algorithm, then some interactions must be left aside to be used in the evaluation. Therefore, it is necessary to split the data before creating a RA.

How the data is split depends on the tested RA, as well as on the technique of evaluation. Different evaluation metrics require different data. For example, one metric requires raw interactions including timestamps and other metadata. In that case, the split can be made for individual interactions:

$$F = F^{train} \cup F^{test} \text{ for } F^{train} \cap F^{test} = \emptyset, \tag{1.22}$$

which means that for the user u and the item i, there may be an interaction at time t_1 that belongs to the training set and another interaction at time t_2 belonging to the test set. This approach is appropriate for RA aimed at recommending items that the user has already interacted with. One example is a model called *reminder*, which recommends the user to buy the same pastry he bought a week ago. However, it is not suitable for evaluation of rating prediction task defined in (1.7), because for $R^{train} = \zeta(F^{train})$ and $R^{test} = \zeta(F^{test})$ with aggregation function ζ defined in (1.5), there is no guarantee that $R^{train} \cap R^{test} = \emptyset$. To properly split the data for evaluation of the rating-prediction task, a constraint must be added to (1.22):

$$F = F^{train} \cup F^{test} \text{ for } \zeta(F^{train}) \cap \zeta(F^{test}) = \emptyset, \tag{1.23}$$

which means that once a f_i interaction for the user u and the item i is added to the test set, then all other interactions between u and i must also be added there. The same goes for the training set.

Besides interactions, users can be split too. This is particularly useful for the evaluation of Top-K recommendations:

$$U = U^{train} \cup U^{test} \text{ for } U^{train} \cap U^{test} = \emptyset. \tag{1.24}$$

Interactions are then assigned to the set in which their author (user) is:

$$F^{train} = \bigcup_{\substack{u \in U^{train} \\ i \in I}} F_{u,i}, \tag{1.25}$$

$$F^{test} = \bigcup_{\substack{u \in U^{test} \\ i \in I}} F_{u,i}. \tag{1.26}$$

All mentioned cases and methods of evaluation will be presented in the following sections. There are also cases where no test interactions are needed to evaluate an algorithm, in which case the RA can be trained on all available interactions and $F^{train} = F$. In all cases, there is the assumption that the set of items I is the same for training and test subset: $I = I^{train} = I^{test}$. The commonly used third set, called the validation set, is intentionally not mentioned here. If a validation set is needed for the RA, e.g., to optimise its hyperparameters, then it can be separated from the training data according to the same rules as the test data.

1.3.1.2 Rating prediction-based evaluation

The RPBE assumes that the recommendation task is defined as a rating prediction task presented in (1.7). RPBE compares the values of predicted ratings $\hat{r}_{u,i}$ with actual ratings $r_{u,i}$ where $r_{u,i} \in R^{test} : r_{u,i} \neq ?$ and R^{test} was split using (1.23). To simplify the denotation in the following definitions, let

$$T = \{(\hat{r}_{u,i}, r_{u,i}) \mid i \in I, u \in U : r_{u,i} \neq ? \land r_{u,i} \in R^{test}\}$$

be a set of ratings that were predicted. The difference between the predicted and actual values is usually measured using one of the following functions [48]. The first and probably most popular function for that is the Mean Square Error (MSE) with a formula:

$$E^{MSE}(T) = \frac{\sum\limits_{\hat{r}_{u,i}, r_{u,i} \in T} (\hat{r}_{u,i} - r_{u,i})^2}{\mid T \mid}.$$

This function is mostly used in optimisation tasks as it is easy to derive it as shown in Section 1.2.3. A modified version called Root Mean Square Error (RMSE) is more commonly used for evaluation:

$$E^{RMSE}(T) = \sqrt{(E^{MSE}(T))}.$$

RSME version is more suitable for evaluation than MSE as it has the same scale as the original ratings and is, therefore, more interpretable. According to [49], RMSE gained popularity for evaluating the RAs used in competitions such as Netflix Price [50] or KDD-Cup 11 [51]. Another function is Mean Average Error (MAE) calculated as

$$E^{MAE}(T) = \frac{\sum\limits_{(\hat{r}_{u,i}, r_{u,i}) \in T} \mid \hat{r}_{u,i} - r_{u,i} \mid}{\mid T \mid}$$

that is less penalising than MSE when the difference is bigger than 1. MAE used to be popular in the early days of RS, but it was replaced by RMSE in around 2006. There is another version of MAE called Normalized Mean Average Error (NMAE) presented in article [52] that solves the problem of comparing results across datasets that have predicated values at different intervals [33]:

$$E^{NMAE}(T) = \frac{E^{MAE}(T)}{\max\limits_{(\hat{r}_{u,i}, r_{u,i}) \in T} r_{u,i} - \min\limits_{(\hat{r}_{u,i}, r_{u,i}) \in T} r_{u,i}}.$$

RPBE approach was used until about 2010 when several experiments showed that minimising RSME may not improve the quality of recommendation algorithms in practical applications. However, Discounted cumulative gain (DCG) and Normalized Discounted cumulative gain (nDCG) remain popular metrics in benchmarks. Since nDCG is designed to measure the quality of ranking, it is also suitable for the RS. The formulas are [53]:

$$DCG_p = \sum_{j=1}^{p} \frac{rel_j}{\log_2(j+1)},$$

$$NDCG_p = \frac{DCG_p}{IDCG_p},$$

where rel_j denotes the relevance of an item that was recommened as j-nth by the RA, p is a number indicating the number of recommended items and $IDCG_p$ is the ideal Discounted cumulative gain:

$$IDCG_p = \sum_{j=1}^{|REL_p|} \frac{2^{rel_j} - 1}{log_2(j+1)}.$$

1.3.1.3 Ranking-based evaluation

The RBE approach is more in line with practical tasks like the Top-K recommendation task described in Section 1.2.2. RPBE focuses on predicting the correct rating for each user-item pair, but real users do not care about the ratings, only about the items recommended to them. The RBE approach respects this fact and, thanks to that, has gained popularity and has become the main offline evaluation approach. By being a Top-K recommendation defined as a binary classification task, when K items should be classified as relevant for the user u, then the commonly used metrics for classification tasks can be used for evaluation of recommendations as well.

A common practice to evaluate binary classification tasks is to create a confusion matrix representing the number of samples classified as positive or negative with respect to whether they are truly positive or negative, as illustrated in Table 1.2.

	classified as positive	classified as negative
truly positive	true-positive (TP)	false-negative (FN)
truly negative	false-positive (FP)	true-negative (TN)

Table 1.2: Confusion matrix for for general binary classification task

There is a list of metrics based on the variables defined in a confusion matrix (TP, FP, FN, TN). The basic two metrics are called recall and precision defined by formulas:

$$precision = \frac{TP}{TP+FP} \qquad \text{and} \qquad recall = \frac{TP}{TP+FN}. \tag{1.27}$$

Recall is also called sensitivity or true positive rate and, as defined in (1.27), is calculated as the portion of samples that have been correctly classified as positive to the number of all positive samples. Precision is calculated similarly as the portion of correctly classified as positive to all classified as positive. The general problem with recall and precision is that they can be greatly affected by the unbalanced representation of classes in the measured set. For example, if the ratio of class *positive* to *negative* was $1 : 99$, the constant model that classifies everything as positive would have a recall equal to 100%, but precision would be 1%. Similarly, if the measured set contains only positive samples and the model classified them as negative in 99% of the cases, then it would still have a precision equal to 100% while recall would be 1%. Neither of these two metrics can guarantee that the classifier is always of good quality. However, together they make a more robust metric called F-score. Specifically, it is a harmonic mean of recall and precision defined as:

$$\text{F-score}_\gamma = (1+\gamma^2)\frac{\mathit{precision} \times \mathit{recall}}{(\gamma^2 \times \mathit{precision}) + \mathit{recall}},$$

where γ is parameter determining the weight of recall. For $\gamma = 1$, the weights of recall and precision are the same. For $\gamma = 2$, recall is twice as important as precision. The most popular setting of γ for F-score is $\gamma = 1$:

$$\text{F-score}_1 = (1+1^2)\frac{\mathit{precision} \times \mathit{recall}}{(1^2 \times \mathit{precision}) + \mathit{recall}} = \frac{2\,\mathit{precision} \times \mathit{recall}}{\mathit{precision} + \mathit{recall}}. \quad (1.28)$$

The evaluation metrics described in Table 1.2 and (1.27) and (1.28) can be used for the binary classification task for which the ground truth is known. However, typically there is no ground truth in Recommender Systems, only interactions. There are several approaches for defining the ground truth for the interactions.

In the case of IF with unary data type such as *purchase* or *detail view*, the approach is straightforward. If the item i has been bought by user u, it is relevant to him. Otherwise, i is not relevant for u. When a dataset contains multiple types of IF, it is possible to distinguish whether i for u is relevant for *purchase*, relevant for *detail view*, etc.. In that case, it is possible to measure quality of RA for predicting particular interaction. Another option is to aggregate interactions, as described in (1.6), and declare i as relevant for u if $r_{u,i} \neq\, ? \wedge r_{u,i} > 0$. The set of relevant items for the user u will be denoted as:

$$RI_u = \{i \mid i \in I : r_{u,i} \neq\, ? \wedge r_{u,i} > 0\}.$$

If the dataset was split according to (1.23), RI_u can contain items relevant according to the test data and not according to training data and vice versa. Therefore, RI_u needs to be split into two subsets:

$$RI_u^{train} = \{i \mid i \in RI_u : r_{u,i} \in R^{train}\},$$

$$RI_u^{test} = \{i \mid i \in RI_u : r_{u,i} \in R^{test}\}.$$

If the dataset has been split by (1.24), then all interactions of u are either training or testing, and there is no need to split the user's relevant items.

For commonly used benchmarks, there are standard approaches to consider item i as relevant for user u. For example, for MovieLens and Netflix, article [54] sets that movie i is relevant for u if u rated i with 4 or 5 stars. In the case of one, two, or three stars, i is considered not to be relevant for u. This approach was adapted by other articles such as [55, 56].

Once the interactions are converted to a binary classification task with relevant (positive) and not relevant (negative) classes, it is possible to measure the listed metrics such as recall, precision, or F1 score for a given RA. The simplest way is to use RA to classify for each user-item pair if an item is relevant for a user and then calculate recall:

$$recall = \frac{\sum\limits_{u \in U^{test}} \mid RA(u) \cap RI_u^{test} \mid}{\sum\limits_{u \in U^{test}} \mid RI_u^{test} \mid}, \tag{1.29}$$

where U_{test} denotes a set of test users that was described in Section 1.3.1.1. However, this method does not take into account the real use of RS as described in Section 1.2.2. For the rest of this section, assume the goal is to evaluate the quality of RA using a recall metric for a Top-K recommendation task.

A completely different approach for measuring recall called *leave-one-out recall* was presented in article [57]. This method is inspired by leave-one-out cross-validation and works as follows: one user is selected and declared as *the active user*. Subsequently, one item relevant to *the active user* is hidden, and the RA's goal is to recommend it in Top-K recommendation task. If the hidden item is included in the K recommended items, the recommendation is classified as successful. This process is repeated for all relevant items of *the active user* and for all users. The *leave-one-out recall* is then calculated as the portion of the number of successful recommendations to the number of recommendations [33], calculated as:

$$recall@K_{LOO} = \frac{\sum\limits_{u \in U^{test}} \sum\limits_{i \in RI_u^{test}} \mid \{i\} \cap Top(K, RI_u^{test} \setminus \{i\}) \mid}{\sum\limits_{u \in U^{test}} \mid RI_u^{test} \mid}, \tag{1.30}$$

where $Top(K, M)$ is a RA that is recommending K items based on items in set M, such as item-KNN algorithm described in Section 1.2.4.

Leave-one-out recall provides a better simulation of the user than (1.29), but still not well enough. It does not follow the user's behavior over time, which is essential for a good offline evaluation, according to [58, 49, p. 262].

When an item is hidden, the RA tries to recommend it using the rest of interacted items. However, some of the items were interacted by the user later than the hidden item, which means that the RS should not know that the user has interacted with them. Thesis [2] noted this drawback and proposed a solution called *leave-last-one-out recall*:

$$recall@K_{LLOO} = \frac{\sum\limits_{\substack{u \in U^{test} \\ (i_1,t_1) \in F_u^{test}}} \mid \{i_1\} \cap Top(K, \{i_2 \mid (i_2,t_2) \in F_u : t_2 < t_1\}) \mid}{\sum\limits_{u \in U^{test}} \mid RI_u^{test} \mid}, \tag{1.31}$$

where F_u is a list of interactions of user u defined as:

$$F_u = \{(i_j, t_j) \mid (u_j, i_j, t_j, v_j) \in F : u_j = u\}$$

and the expression

$$\{i_2 \mid (i_2, t_2) \in F_u : t_2 < t_1\}$$

represents the set of items that were interacted by the user u before timestamp t_1. The benefit of the *leave-last-one-out recall* is that it uses only interactions that were available at the time when the hidden item should have been recommended.

Thesis [33] mentions another drawback of *leave-one-out recall*, this time from a business perspective. Users with many interactions have much higher weight when computing recall than those with a few interactions. This causes, among other things, considerable susceptibility to the presence of robots and crawlers in the data. Once a robot with a huge number of interactions appears in the data and is not properly filtered out, the measured leave-one-out recall is misleading. Thesis [33] suggests that all test users should have equal weight and defines the formula:

$$recall@K_{LOO}^{UN} = \sum_{u \in U^{test}} \frac{\sum\limits_{i \in RI_u^{test}} \mid \{i\} \cap Top(K, RI_u^{test} \setminus \{i\}) \mid}{\mid RI_u^{test} \mid}, \tag{1.32}$$

which is called *user-normalised leave-one-out recall@K*.

However, the evaluation of RA for Top-K recommendation task using *leave-one-out* or *leave-last-one-out recall* has a major drawback with regard to the long-tail effect. In order to maximise (1.29), it is far more important to correctly recommend the popular items at the expense of the unpopular ones. As a result, the RA optimised using this metric may recommend only a few of the most popular products. This means that algorithms optimised using (1.30) or (1.31) only magnify the impact of the MNAR problem (see Section 1.1). It is natural that some items are more popular than others, according to a number of interactions, but this could be due to something completely different than being really relevant to more people. For example, the e-commerce platform ran a huge advertising campaign for the item i, a lot of people bought i, making i the most popular item of all. Subsequently, the recommendation algorithm is optimised on this data, and the algorithm learns that everyone wants the item i and no longer takes into account other items that might be more appropriate for them, but they were not that advertised at the beginning. The existence of the bias was experimentally verified in article [59]. One way to partially address the problem with bias in the data is with an appropriate sampling strategy [60]. Article [61] noted this problem and offered a solution called *popularity-stratified recall.* According to it and [33], popular items should be penalised with a formula:

$$recall_{PS}^{\beta} = \sum_{u \in U^{test}} w^{\beta}(u) \frac{\sum\limits_{i \in RI_u^{test} \cap RA(u)} p(i)^{-\beta}}{\sum\limits_{i \in RI_u^{test}} p(i)^{-\beta}}, \tag{1.33}$$

where $p : I \to [0,1]$ denotes relative item popularity computed as

$$p(i) = \frac{\sum\limits_{u \in U^{train}} \mid F_{u,i} \mid}{\sum\limits_{j \in I} \sum\limits_{u \in U^{train}} \mid F_{u,j} \mid}$$

and $\beta \in [0,1]$ is the parameter that determines how much popular items should be penalised and $w^{\beta}(u) \in [0,1]$ is a weight of user u. Sum of weights for all users must sum up to one, $\sum\limits_{u \in U^{test}} w^{\beta} = 1$, with suggested:

$$w^{\beta}(u) = \frac{1}{\mid U^{test} \mid} \frac{\sum\limits_{i \in RI_u} p(i)^{-\beta}}{\sum\limits_{v \in U^{test}} \sum\limits_{i \in RI_v} p(i)^{-\beta}}.$$

It is interesting to note that for $\beta = 0$ all the $p(i)^{-\beta}$ are equal to one and (1.33) and (1.29) are the same.

1.3.1.4 Other standard methods

In addition to recall and other metrics listed in the previous section, there is one more commonly used metrics called *catalog coverage* (CC). However, CC does not measure the success of recommendations, but calculates a portion of recommended items to all items, which means it is an indicator how many items is the recommendation algorithm capable of recommending [5]. For the set of test users U^{test} and $RA(u)$ returning set of relevant items for user u, CC is defined as:

$$catalog{-}coverage = \frac{\bigcup\limits_{u \in U^{test}} RA(u)}{\mid I \mid}.$$

Besides CC, there are metrics dedicated to the evaluation of RS in specific use cases. For example, [58] also mentions other theoretically described metrics such as Novelty, Confidence, Trust, Serendipity, and Diversity.

1.3.2 Online evaluation

Another approach to evaluate a RA is based on the feedback of real users. Specifically, items are recommended to the real user, and the RS collects the user's reactions to the recommended items. An example of a reaction is when the user clicks on the recommended item. The reactions observed depend on the domain of the platform. For example, for a web search domain, feedback called *time to click on search result page* is measured [62].

The evaluation of the quality of the RA is based on the collected feedback. As mentioned in Section 1.1, feedback can be implicit or explicit. Both types of feedback can be used for online evaluation, although metrics based on IF are more widely used. Metrics based on explicit feedback also exist but are less common. The most commonly used metric based on the IF for its universality is the click-through rate (CTR). Article [63] claims that the CTR measures how often recommendations are accepted by users. Therefore, the CTR can be calculated as:

$$CTR = \frac{\text{number of accepted recommendations}}{\text{number of recommendations}}.$$

The recommendation was accepted by the user if he clicked on at least one recommended item. The fact that a user has clicked on a recommended item must be made explicit to the Recommender System. If the Recommender System has explicit information that the user clicked on an item as a result of the recommendation, then an explicit CTR is measured. If the Recommender System does not have explicit information that the item has been clicked through based on the recommendation, then the implicit CTR can be measured based on interactions. Specifically, it is observed whether a user interacted with a recommended item immediately after a recommendation. Formally:

$$REC : Z_t \times U \times \{0,1\}^I$$

defines a set of recommendations where each recommendation is represented by a timestamp $t \in Z_t$, user $u \in U$ and by a set of recommended items $I' \subset I$:

$$rec_j = (t_j, u, I').$$

When assuming that the set of interactions F contains only detail view interactions, then the implicit CTR (iCTR) can be calculated as:

$$iCTR(d) = \frac{\sum\limits_{(t,u,I') \in REC} sgn(\mid I' \cap F_u(t,d) \mid)}{\mid REC \mid}, \tag{1.34}$$

where $F_u(t, d)$ is a set of items interacted by the user u within time $[t, t + d]$ defined as

$$F_u(t, d) = \{i_j \mid (i_j, t_j) \in F_u : (t_j >= t) \wedge (t + d >= t_j)\},$$

where d is a parameter determinating how long after the recommendation the user has to interact with the recommended item to mark the recommendation as successful. Other metrics based on other types of interactions can be defined in a similar way. For example, a cart conversion rate (CCR) measures what proportion of a recommendation is followed by the addition of items to a cart, or a conversion rate (CR) is defined as the proportion of the recommendation that led to the purchase of an item.

The CTR is an example of a short-term reward that can be part of a long-term reward (e.g. sales) [64]. The downside of the CTR is that it is significantly influenced by other factors and not just the quality of the RA [65]. Articles [66, 67] present a large number of factors that influence whether a user notices a recommendation. An example of these external factors is the layout of the page or the format in which the recommendation is presented. For example, if a list of recommended items is presented in the footer of a long page, the RA's CTR will be lower than for the recommendations presented as the main content of the page. Therefore, a CTR cannot be used to obtain an exact number indicating the quality of the recommendation algorithm. However, it is ideal for comparing two or more algorithms using an A/B test.

The A/B test is a common way to compare different versions of the system (web applications, desktop applications, Recommender System) by dividing users into groups. Each user group is presented with a different version of the system, and the A/B test evaluates which user group has responded better to their version of the system [45]. In the case of the Recommender System, the different versions of the system are represented by different RAs. At the beginning of the A/B test, it is necessary to define the different versions of the system (for example, different parameters of the RA) and their proportion of users. At the end of the A/B test, an assessment is made of which group of users have the largest CTR, and the corresponding recommendation algorithm is declared the best. The A/B test requires considerable engineering effort [62]. One of the tasks the A/B test administrator must address is to assign new users to individual groups to maintain the desired ratio of users between groups. It is also necessary to be aware of robots, scrapers, and other nonhuman users, as they can significantly affect the results of the A/B test [68]. A/B tests are also suitable for testing hypotheses about RAs. The length of the A/B test is then typically determined by the number of users required to reach statistically significant results.

1.3.3 Comparison of online and offline evaluation metrics

Offline and online metrics such as recall and CTR have been described in Sections 1.3.1 and 1.3.2. Now, these metrics will be compared.

The main advantages of offline evaluation are that it is simpler to implement, repeatable, fast and can compare any number of models. The downside of offline evaluation is that it does not describe real RS, as well as online evaluation [69]. Another problem with offline evaluation is the bias in the interactions described in Section 1.3.1.3. Therefore, offline evaluation should be used to find a few well-functioning RAs to be subsequently compared using online evaluation [58, p. 261]. A similar process is described in [70, p. 401], where an offline evaluation of the RA is performed to test the hypothesis, and if the hypothesis is not disproved, the RA is tested using an A/B test.

The online evaluation of the RA is also better from a business perspective, as it can take into account different business rules (e.g., the requested diversity of recommended items) [70, p. 401]. The downside of online evaluation is the required time to get results. In the case of offline evaluation, the only resource required is computing power, which can be easily obtained if needed to speed up the evaluation. For online evaluation, the most valuable resource is the real users, who are typically available in limited numbers. Therefore, an online evaluation of RA using an A/B test may take several weeks to collect the necessary data. It also means that the results of an online evaluation are unreproducible.

There are only a few (compared to offline evaluation) articles describing online evaluation since it takes real users working with a live Recommender System to make an online evaluation. However, there are some articles comparing online and offline metrics. Probably the most interesting article is [71], which compares offline and online metrics on the Swiss news website *swissinfo.ch*. Specifically, they optimise RAs based on their own offline metric called *success*@K and measure the CTR of the created RAs further in the online environment. As a result, RAs dominating according to offline metrics are no longer as good in the online environment because they favour popular items. In contrast, the RA recommending random items comes out far better in online evaluation than offline because it helps a user to explore content.

Article [49] seeks to combine online and offline evaluation and describes how to measure recall in the online environment using real users. It refers to a technique called *prequential evaluation* described in [72] and notes that it can be used for content that changes very frequently (such as streaming platforms). The work also mentions that the difference between offline and online evaluation is not clear and defines that offline evaluation is any evaluation computed using static data and local computer, while online evaluation is done in real RS.

Chapter 2

Proposed experiment

In Chapter 1, Recommender Systems were introduced with emphasis on different types of interactions, matrix factorisation, and, most importantly, evaluation. At the end of Section 1.3.3, the advantages and disadvantages of offline and online evaluation were described. In Section 1.3.1.3, problems with the recall metric were pointed out and more options how to calculate recall were presented. Specifically, the following versions of recall were defined:

- *leave-one-out recall* defined in (1.30) measuring the success of Top-K recommendation algorithm using leave-one-out validation,
- *leave-last-one-out recall* defined in (1.31) measuring the success of Top-K recommendation algorithm using leave-one-out validation considering the sequential nature of interactions,
- *user-normalised leave-one-out recall* defined in (1.32) for penalising active users and as protection against robots,
- *popularity-stratified recall* defined in (1.33) with parameter β penalising popular items to remove bias in data.

The chapter will present the methodology of the experiment proposed in this work. The individual steps of the experiment will be described in detail, including an emphasis on the issues that were solved. In addition, other versions of recall created as a combination of already described versions of recall will be introduced.

2.1 Hyperparametrization of recall metric

As mentioned, there are doubts that maximising recall also maximises CTR. This work aims to explore more deeply the relation between recall and CTR in real applications. Different versions (hyperparametrizations) of recall will be measured, and it will be determined which version of recall is the best for evaluating the RA to maximise the CTR. Specifically, the versions of recall proposed and examined are:

- *popularity-stratified user-normalised leave-one-out recall* (PS-UN-LOO recall) defined by (2.1),
- *popularity-stratified user-normalised leave-last-one-out recall* (PS-UN-LLOO recall) defined by (2.2).

Both proposed versions of recall have hyperparameter β determining how much popular items should be penalised. Furthermore, (2.1) and (2.2) differ only in the method of cross-validation, where the leave-last-one-out version takes account of the sequence of interactions as described in Section 1.3.1.3. Settings of the cross-validation method can also be taken as a hyperparameter with values

$$VAL \in \{\text{leave-one-out (LLO)},\ \text{leave-last-one-out (LLOO)}\}. \tag{2.3}$$

Another hyperparameter is K expressing the number of recommended items in the Top-K recommendation. Including the parameter K is slightly speculative since K is a setting of the RA and not the evaluation metric. However, it will also be included in the experiment as it may show a link between the order of the recommended item and its click-through rate. During the experiment, it will be determined how adjusting of K, VAL, β affects the value of recall.

2.2 Experiment steps

The idea behind the experiment is to create several different models (RAs), deploy them in a real RS, and measure their recall (for individual hyperparameters) and CTR. The proposed experiment consists of the following steps:

1. First, there are several models to be found that will vary in performance (measured using recall). An item-KNN algorithm with the similarity of items measured by the cosine similarities of ALS embeddings was chosen as the model. As described in Thesis [2], the performance of this model depends on its hyperparameters. Specifically:
    - *factors* - number of dimensions of the latent space,
    - λ - regularisation parameter used in (1.18),
    - $BM25_b$ - parameter of BM25 weighting,
    - *pruning-item* - the minimum number of items a user had to interact with to avoid being filtered out of the rating matrix,
    - *pruning-user* - the minimal number of users who had to interact with an item in order to not be filtered out.

$$recall@K^{\beta,UN}_{LOO,PS} = \sum_{u \in U^{test}} w^{\beta}(u) \frac{\sum\limits_{i \in RI^{test}_u} \mid \{i\} \cap Top(K, RI^{test}_u \setminus \{i\}) \mid \ p(i)^{-\beta}}{\sum\limits_{i \in RI^{test}_u} p(i)^{-\beta}} \tag{2.1}$$

$$recall@K^{\beta,UN}_{LLOO,PS} = \sum_{u \in U^{test}} w^{\beta}(u) \frac{\sum\limits_{(i_1,t_1) \in F^{test}_u} \mid \{i_1\} \cap Top(K, \{i_2 \mid (i_2, t_2) \in F_u : t_2 < t_1\}) \mid \ p(i)^{-\beta}}{\sum\limits_{i \in RI^{test}_u} p(i)^{-\beta}} \tag{2.2}$$

This step of the experiment involves exploring a part of the space of these hyperparameters to find hyperparameters of models with different performances.

2. Once hyperparameters are found for several models with different values of recall, a further step can be taken. The next step is splitting the users into a training and test set. Because recall will be measured on the test set, users assigned to the test set have to have at least two interactions. On a user with only one interaction, it is not possible to measure recall of item-KNN algorithm using the LOO or LLOO technique. Due to the emphasis on the precision of the measured recalls, there is an effort to set the size of the test set of users larger than it is normally used. Specifically, the experiment was designed so that the set of test users was either 10% of all users with a maximum of 100,000 users. The split of users into training and test is done once before the next step as it simulates the standard approach of measuring recall on static data.

3. Once the test users are separated, it is possible to train the models on the rest of the users. These models, which will be called production models, differ from the normal models created in the first step of the experiment. The first specificity of the production model is that it is automatically deployed to a real RS to recommend for real customers, and thus its CTR can be measured. The second specificity is that the model is constantly receiving new interactions from customers and it is re-trained periodically to include them. Interactions may also come from new users who have not been in a source rating matrix before. New users are added to the set of training users, and their interactions are used for creating production models. The test set of users remains unchanged. Constantly re-training the model is a key part of the experiment because, without re-training, the model could quickly become obsolete, with negative consequences for CTR and thus business. In addition, the experiment would need to take into account how quickly the model becomes obsolete on the selected domains. For example, the model would become obsolete far more quickly for the platform with online news than for e-commerce with a fixed assortment. By periodically updating the production model, the proposed experiment can be carried out for any length of time without negative consequences for a business.

4. Created production models that generate recommendations for real customers are also used in the next step to measure recall on the set of test users. But since production models are recalculated periodically, recall has to be measured repeatedly as well. Specifically, recall must be measured again when the production model is changed (updated), and the production model can be changed only when recall was already measured. Therefore it is necessary to implement a non-trivial mech-

anism to ensure the synchronisation of these two parallel processes. A description of the implementation will be given in Chapter 3. With repeated measurements of recall, it is also necessary to take into account new interactions of recommended items because new interactions change the popularity of items $p(i)$ and, as a result, have an impact on recall as described by (2.1) and (2.2).

2.3 Description of experimental data

Once the individual steps of the experiment are fulfilled and implemented, the experiment is performed on several datasets. Result of the experiment is, among other things, a sequence of measured CTRs along with the number of users (nu) that interacted with the model during the individual time periods:

$$\begin{aligned} s_l^A = \{ & \\ & (CTR(t_1, t_2, m_{l,1}^A), nu(t_1, t_2)) \\ & (CTR(t_2, t_3, m_{l,2}^A), nu(t_2, t_3)) \\ & \dots \\ & (CTR(t_o, t_{o+1}, m_{l,o}^A), nu(t_o, t_{o+1})) \\ & \}, \end{aligned} \tag{2.4}$$

where model m_l^A is one of the production models for dataset A defined by hyperparameters listed in the first step of the experiment and (t_1, t_2) indicates the time period when the $m_{l,1}$ model was a production model and therefore recommended to real users. For each dataset, there are L sequences like this, since $l \in \{1, \dots, L^A\}$ where L^A is the number of models deployed for real users from dataset A. Recall is also measured for each deployed model for a list of hyperparameters K, VAL, β:

$$\begin{aligned} e_l^A = \{ & \\ & recall@K_{VAL,PS}^{\beta,UN}(t_1, t_2, m_{l,1}^A) \\ & recall@K_{VAL,PS}^{\beta,UN}(t_2, t_3, m_{l,2}^A) \\ & \dots \\ & recall@K_{VAL,PS}^{\beta,UN}(t_o, t_{o+1}, m_{l,o}^A) \\ & \}. \end{aligned} \tag{2.5}$$

Both sequences are a good source of data to observe changes in CTR and recall over time, as well as to study the impact of recall's individual hyperparameters. For further analysis to determine the CTR and recall relation, s_l^A and e_l^A sequences are needed to be aggregated. In the case of recall, just calculate a simple average, i.e.:

$$E_l^A = \frac{1}{o} \sum_{j \in \{1,\dots,o\}} e_{l_j}^A. \tag{2.6}$$

In the case of CTR, the number of users involved in the measuring of CTR should also be taken into account. Each period is about the same length and is equal to the time needed for recall measurement, with the result that s_l^A sequences include the CTR of users at different times of the day. For example, $s_{l_4}^A$ can be measured between 2 a.m. and 4 a.m., when the number of users is very low compared to daily traffic. Therefore, it is appropriate to calculate a weighted average when calculating the average CTR, where the weight is set by the number of users:

$$S_l^A = \frac{\sum_{j \in \{1,\dots,o\}} CTR(t_j, t_{j+1}, m_{l,j}^A)\ nu(t_j, t_{j+1})}{\sum_{j \in \{1,\dots,o\}} nu(t_j, t_{j+1})}. \tag{2.7}$$

Subsequently, the vector of average CTRs can be taken for dataset A and its L models:

$$S^A = (S_1^A, S_2^A, \dots, S_L^A)$$

and similarly for the vector of average recall values:

$$E^A = (E_1^A, E_2^A, \dots, E_L^A).$$

One E^A vector is measured for each combination of hyperparameters VAL, β and K. Until now, these hyperparameters have not been explicitly written to clarify the notation, but now they will be. Therefore, the entire list of vectors of recall is measured according to their hyperparameters:

$$EL^A = \{E_{VAL_1,\beta_1,K_1}^A, E_{VAL_2,\beta_2,K_2}^A, \dots, E_{VAL_V,\beta_V,K_V}^A\}, \tag{2.8}$$

where V is the total number of hyperparameter combinations and VAL_j, β_j, K_j are the individual hyperparameter values. A Spearman correlation coefficient (ρ) is computed between each hyperparameterization of vector of average recall values E_{VAL_j,β_j,K_j}^A and the vector of average CTRs S^A:

$$corr_j^A = \rho(E_{VAL_j,\beta_j,K_j}^A, S^A). \tag{2.9}$$

(2.9) results in a correlation coefficient indicating whether there is a monotonically increasing function between the recall values and CTRs measured on dataset A. To determine whether the obtained results are domain-dependent, multiple datasets are included in the experiment. The set of tested hyperparameters of recall must be the same for all datasets. Analogically, there will be $corr_j^B$, $corr_j^C$, etc.

Chapter 3

Implementation

The chapter will first present the datasets used in the conducted experiment. Following the size of the selected datasets as well as the existing software solution to collect user feedback, the implementation of the experiment will be presented. The method of synchronising the production model with the measurement of recall, as already outlined in Section 2.3, will be explained. A large number of optimisations that were needed to be done will be mentioned.

3.1 Datasets

Commonly used datasets for research of RS such as *MovieLense* or *Last.fm* cannot be used in the experiment since real users are required for online evaluation. Because of this, the work was created thanks to the generosity of SaaS RS called Recombee using their customers. Specifically, the customers described in Table 3.1 were selected.

There are four types of collected implicit interactions for the listed datasets. Their description and assigned values as described in (1.3) are:

- an item was viewed in detail with assigned value $v = 0.25$,
- an item was added to a cart with assigned value $v = 0.5$,
- an item was bookmarked with assigned value $v = 0.5$,
- an item was purchased with assigned value $v = 1$.

3.2 Modules

The experiment is implemented as a series of modules that can be connected to the internal interface of the production RS. The vast amount of data to be processed is taken into account in the implementation. Emphasis is placed on

Name	Description	# Interactions	# Users	# Items	Recommendations per day
Dataset A	Philippine liquor e-shop	2.8m	1.3m	3k	6 - 12k
Dataset B	English pet store	16.8m	6.9m	20k	75 - 115k
Dataset C	Brazilian fashion e-shop	30.4m	3.1m	19.3k	6 - 8k
Dataset D	B2B supplier of African goods	13.9m	70k	1k	15 - 40k
Dataset E	Dutch videostream service	30.7m	1.8m	5.3k	16 - 80k

Table 3.1: Description of datasets

efficient storage of interactions as well as on the implementation of the measurement of recall. The Python language was chosen for the implementation of the module, so it can cooperate with already existing modules of the used RS.

The modules and interfaces of the existing RS at the start of the experiment were:

- The **Factorisation** module is responsible for computing matrix factorisation, resulting in ALS embeddings. On input, it is given a hypeparametrization described in the first step of the proposed experiment described in Section 2.2, and the output is an embedding for each item. It runs in the background and periodically accepts new users and recalculates ALS embeddings.

- The **EmbeddingSimilarityComputer** (ESC) is responsible for calculating the NNs for each item. The metric of similarity is the cosine similarity defined in (1.8). The input is a set of embeddings of all items, and the output is a dictionary containing a list of K nearest neighbours along with the distances to them.

- The **BasicStorage** (BS) is a connector to a database that contains information about users, items, interactions, and other necessary tables used by the Recommender System.

- The **Worker** is the component responsible for creating recommendations. The input to the component is a table of nearest neighbours created using ESC, and the output is recommendations for individual users created using the item-KNN algorithm.

These modules were used in the experiment as separate modules or as part of other modules that were developed for the experiment. Specifically, the following classes have been implemented:

- The **InteractionPreparator** loads interactions from the database using **BS** on-demand and contains a caching mechanism to effectively store them in the memory for future requests. It is also responsible for clearing data from robots, pruning them to eliminate noise, or aggregating them into an interaction matrix.

- The **GSheetLogger** is for logging results into Google Sheet. It was developed to find the right hyperparameters for the ALS method. It is possible to monitor the progress of a calculation that takes several days, the intermediate results of it, and in the end, the effect of hyperparameters on the result. Its saved data is also used to restore the calculation after an unexpected application crash.

- The **RecallMeter** is the most important class implemented in this thesis. It contains highly optimised methods capable of measuring recall for large amounts of hyperparametrization at once. For example, for the requirement to measure recall defined by (2.2) for the Cartesian product of many values of hyperparameter K with many values of hyperparameter β, it takes only one request to RA. The hardest part was implementing a method that allowed multiple users to be evaluated in parallel. Most of the calculations were vectorised, but for some parts, parallel methods of calculation had to be used, which are very problematic in Python because of GIL. A number of parallel approaches have been tried, resulting in the use of the *pandarallel* library. Because of the enormous size of the set of test users, the parallelization of the calculation was absolutely crucial, and the experiment would have been far less accurate in its measured values without it.

The classes described are used in four modules. Each runs in the background as an own application. These are:

- The **DataSplitter** module is responsible for splitting users into predefined sets (such as training and test subsets). It can also take account of new users of the RS. It uses an instance of the **InteractionPreparator** to load interactions and then performs a split of users according to specified criteria (for example, the test user must have at least two interactions). The user split is saved using **BS** to the database.

- The **AutoALS** module aims to search the hyperparameter space and find high-quality but different ALS embeddings for the experiment. It loads interactions using an instance of **InteractionPreparator** and logs the progress of the search using a **GSheetLogger** instance. To evaluate the created embeddings, **ESC** and the instance of **RecallMeter** are used with the settings $\beta = 5$, $VAL =$ leave-last-one-out and $K = 5$. It is optimised to cache intermediate results for faster search of the space.

- The **MeasureRecall** module is a wrapper around an instance of **RecallMeter**. Its responsibility is to load the table of nearest neighbours with **BS** and prepare the interactions of test users with **InteractionPreparator**. It also makes sure to measure recall for all currently used models, which means he has to know when the production model changed and starts measuring recall for it. It also needs to ensure that this model is not overwritten until recall is measured. This produces pairs of CTR and recall corresponding to the same model as described in Steps 3 and 4 of Section 2.2. Once recall is measured, the model is marked as obsolete and can be overwritten with a new one. In the meantime, the module is measuring recall of the next model.

The implementation of the listed classes and modules is attached to this thesis. The code follows the PEP 8 Style Guide for Python and contains documentation according to NumpyDoc Style Guide.

The overview and communication of these modules can be seen in Figure 3.1. In the beginning, there is a search for hyperparameters of ALS embeddings, which took about a week for one dataset. Once the hyperparameters have been found, a series of calculations are triggered, constantly running in infinite loops. Specifically, it is:

1. **Factorisation** calculates new ALS embeddings,
2. **ESC** takes embeddings and finds nearest neighbours,
3. **MeasureRecall** evaluates the model represented by a table of nearest neighbours, saves recall and informs **ESC** that it can already rewrite the table of nearest neighbours for a particular model.

Individual modules can run independently on an own computing server. The longest-lasting operation was the recall measurement due to the vast number of hyperparameters tested and the abnormally large number of test users. Therefore, most of the cores were assigned to the **MeasureRecall** module. Because all modules used for one dataset fully stressed all threads of the computing server, each dataset was processed in parallel on its own server. Specifically

- 2x server with 256GB RAM and Intel Xeon E5-165 (12 threads, 3.5GHz),
- 3x server with 64GB RAM and Intel Core i7-7700 (8 threads, 3.6GHz)

were used in the experiment. Since the individual modules were deployed as separate processes, they had to communicate using an external resource, such as the database. Using **BS** a table is created and individual modules write into it in producer-consumer format. For example, **ESC** writes that it has produced a new table of its nearest neighbors and knows that it cannot rewrite it until it has been consumed by the **MeasureRecall** model. The same applies to **Factorialisation** and **ESC** in case of ALS embeddings.

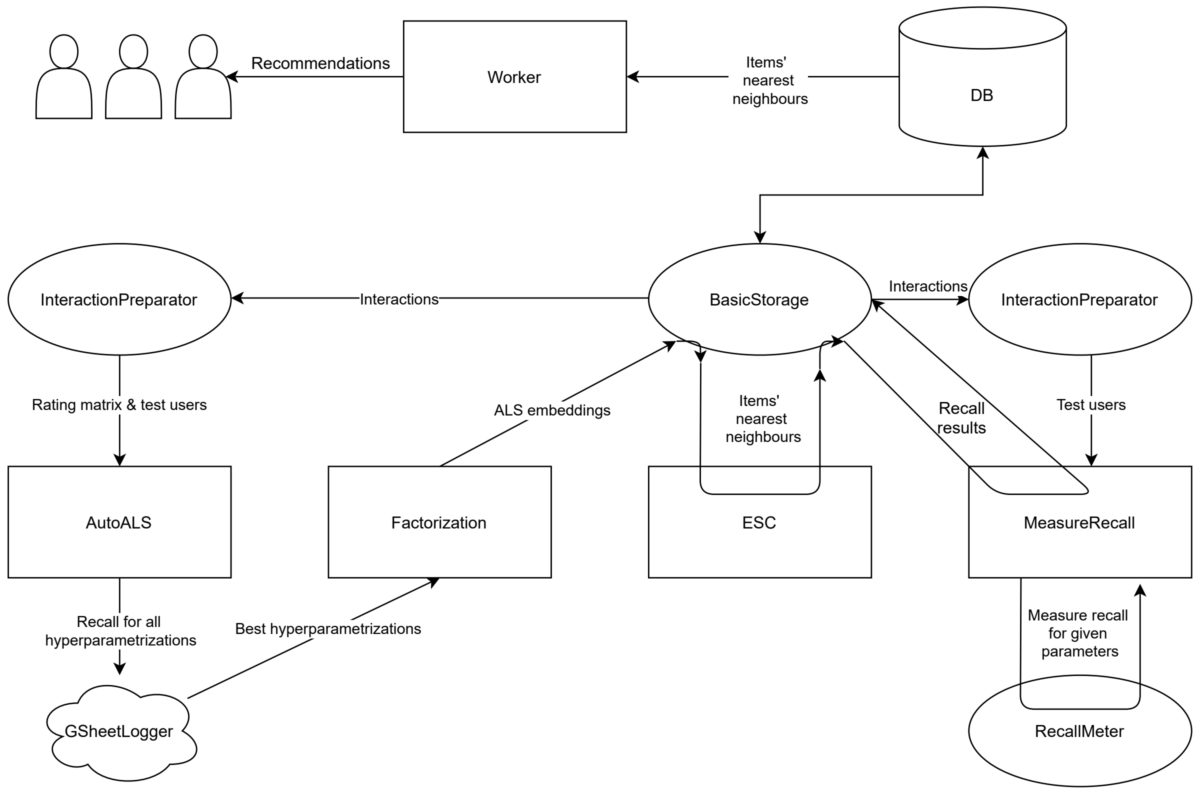


Figure 3.1: Overview of the experiment

Chapter 4

Results

First, the chapter will present the values of the hyperparameters of ALS embeddings described in Section 2.1 and then the tested values of hyperparameters of recall measurement in Section 2.2. The data collected in the experiment will be presented then. For a better understanding of the studied metrics, the evolution of CTR and recall over time will be plotted and described, as well as the impact of the hyperparameters on the measured recall. The chapter will be completed by analysing the relation of recall and CTR, as described by (2.4)-(2.9).

4.1 Hyperparameters

In Section 2.3, the L^A variable was defined as the number of models used for recommendation to real users in dataset A. For the experiment, the values were set to $L^A = L^B = L^C = L^D = L^E = 5$, so the first step of the experiment looked for five hyperparameterization of ALS embeddings. Hyperparametrizations found are illustrated in Table 4.1 along with the labels of the individual models in the *Notation* column.

The RS split users for each model equally. Once the user was assigned to the model during the first recommendation, then any further recommendation was generated by the same model.

The values of the β and K hyperparameters for which recall was measured during the test were:

$$\beta \in \{0.05j \mid j \in \{0, 1, \ldots, 20\}\},$$

$$K \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 50\}.$$

The VAL hyperparameter described in (2.3) containing two values was introduced for the cross-validation method. A Cartesian product was used to get the β_i, VAL_i, K_i triples as described in (2.8) resulting in 588 different versions of recall measured for each model in Table 4.1.

Dataset	Notation	factors	λ	$BM25_b$	pruning-user	pruning-item
A	M_{A1}	32	0.1	0.75	2	5
A	M_{A2}	128	0.1	0.75	2	5
A	M_{A3}	256	0.1	0.75	2	5
A	M_{A4}	256	0.1	0.75	5	5
A	M_{A5}	1024	0.1	0.75	2	5
B	M_{B1}	32	1.0	0.75	5	5
B	M_{B2}	128	1.0	0.75	2	15
B	M_{B3}	256	1.0	0.75	2	20
B	M_{B4}	1024	1.0	0.75	2	20
B	M_{B5}	1024	1.0	0.75	3	10
C	M_{C1}	32	1.0	0.75	2	5
C	M_{C2}	64	1.0	0.75	2	5
C	M_{C3}	128	1.0	0.75	10	2
C	M_{C4}	512	1.0	0.75	5	2
C	M_{C5}	1024	10.0	0.75	5	2
D	M_{D1}	32	0.1	0.75	2	2
D	M_{D2}	64	0.1	0.75	2	2
D	M_{D3}	128	0.1	0.75	2	2
D	M_{D4}	256	0.1	0.75	2	2
D	M_{D5}	1024	10.0	0.75	2	2
E	M_{E1}	32	10.0	0.75	2	2
E	M_{E2}	64	0.1	0.75	20	2
E	M_{E3}	256	1.0	0.75	2	2
E	M_{E4}	1024	1.0	0.75	10	2
E	M_{E5}	1024	10.0	0.75	2	2

Table 4.1: Hyperparameters of ALS embedding for used models

4.2 Collected data

The time taken to measure all versions of recall is various for each model. A number of things have affected the calculation, such as the number of recommendable items, the amount of interaction of test users, or the number of server cores. The longer it took to measure recall, the more obsolete the model became because it could not be recalculated to take account of new interactions. Therefore, the number of measurements of recall (denoted as o in Section 2.3) differs for each model. For each model, CTR was also measured, namely, the implicit CTR (iCTR) defined in (1.34) with a parameter $d = 10$. The number of users and the number of their recommendations participating in the iCTR depends on the length of recall measurement, dataset and other external circumstances such as the time of day. The measurements took place over 18 days. All these values are shown in Table 4.2.

Model	o	duration of one iteration [hh:mm]	iCTR # users	iCTR # interactions
M_{A1}	28	14:43	7.012	29.444
M_{A2}	29	14:31	7.122	14.767
M_{A3}	28	14:45	6.986	13.650
M_{A4}	28	14:43	7.117	34.821
M_{A5}	28	14:42	7.102	20.622
M_{B1}	39	10:44	70.155	133.462
M_{B2}	38	10:43	67.147	127.785
M_{B3}	39	10:44	70.061	210.716
M_{B4}	39	10:43	68.680	130.597
M_{B5}	39	10:43	67.749	128.969
M_{C1}	12	33:51	6.522	19.881
M_{C2}	11	36:02	6.341	19.875
M_{C3}	11	34:57	6.279	18.946
M_{C4}	11	34:56	6.028	18.242
M_{C5}	12	33:17	6.106	18.454
M_{D1}	12	32:22	19.529	66.680
M_{D2}	12	32:22	20.023	67.615
M_{D3}	13	30:57	19.103	66.836
M_{D4}	12	32:20	20.253	71.063
M_{D5}	12	32:22	17.939	62.907
M_{E1}	8	50:14	21.106	75.974
M_{E2}	8	48:07	19.862	71.941
M_{E3}	7	48:11	16.503	56.311
M_{E4}	8	45:42	19.875	71.642
M_{E5}	9	45:38	20.853	72.628

Table 4.2: Hyperparameters of ALS embedding for used models

A large amount of data will now be analysed. In particular, the development of the CTR over time will be shown first, then the effect of the hyperparameters on recall will be shown, and finally, the relation between recall and the CTR will be analysed.

4.3 Analysis of CTR

The CTR is generally an insufficiently theoretically researched metric compared to recall. For example, common sense would suggest that the CTR will not change significantly over time. The model can at most become obsolete, and thus the quality of its recommendations deteriorate. However, the period appears in the data collected, as seen in Figure 4.1 for dataset A and in Figure 4.2 for dataset B. If these periodic changes were due to model obsoles-

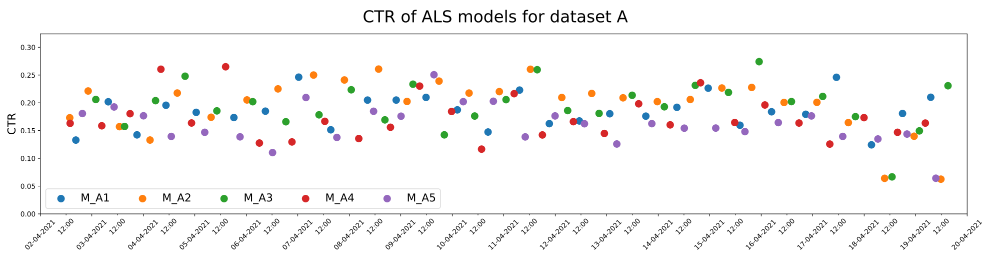


Figure 4.1: Changes of CTR over time for dataset A

cence, the CTR should increase after a model update. However, that is not happening. After the update, the CTR gets lower or higher depending on the time of day.

The explanation for this phenomenon is not straightforward and is likely to be domain-dependent. It can be assumed to be related to user behaviour at certain times of the day. Dataset A shows about a 1-day period in the CTR, which could mean that users who visit an e-shop with alcohol during the day behave differently from night visitors. The highest CTR is measured for the model operating in the afternoon (UTC), which represents the late evening in the Philippines. It means that late-evening customers of the e-shop are more likely to click on recommended items than morning users.

Similarly, for dataset B containing interactions from an English pet store, the daily period can be seen. The interesting thing about dataset B is the increase in recall of all models that occurred around 08/04/2021. Based on the collected data, it is not possible to estimate with certainty the reason for this phenomenon. For example, a potential reason could be to change the form the recommendation was presented in, as described in Section 1.3.2. Another reason for such a rapid and significant change could be the social events in Britain caused by the coronavirus, such as the new regulations.

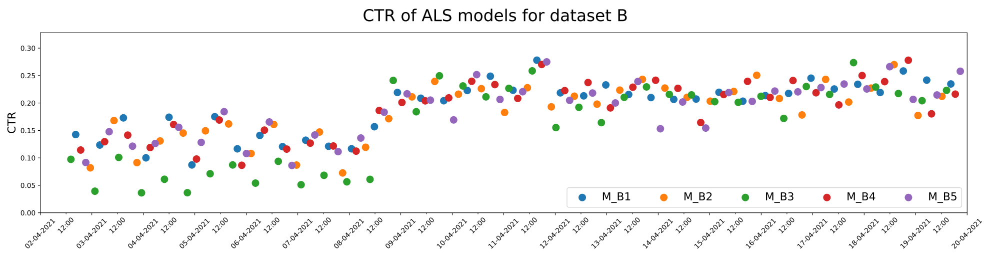


Figure 4.2: Changes of CTR over time for dataset B

The progress of the CTR measured on dataset C is shown in Figure 4.3. Dataset C, the e-shop with T-shirts and other clothes, shows no signs of any periodical changes. Still, the CTR can be seen to change over time and may increase by as much as 50% in a few days for no apparent reason.

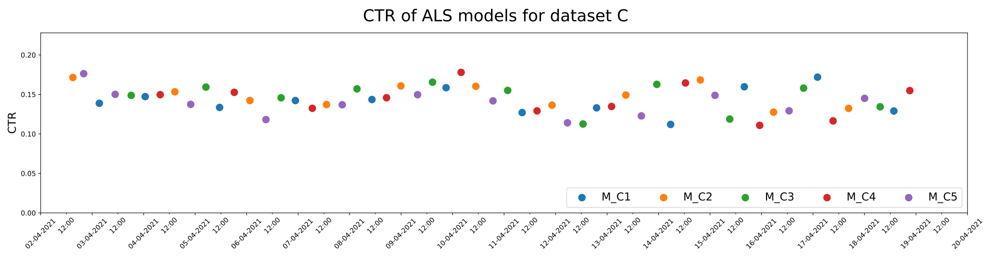


Figure 4.3: Changes of CTR over time for dataset C

Figure 4.4 shows the CTR of B2B supplier of African goods represented by dataset D. It follows a similar trend to dataset B but with a weekly and not a daily period. The highest CTR is achieved on weekends, which is a little surprising because the customers are companies. It can be assumed that the companies (users) are looking for goods for the following week.

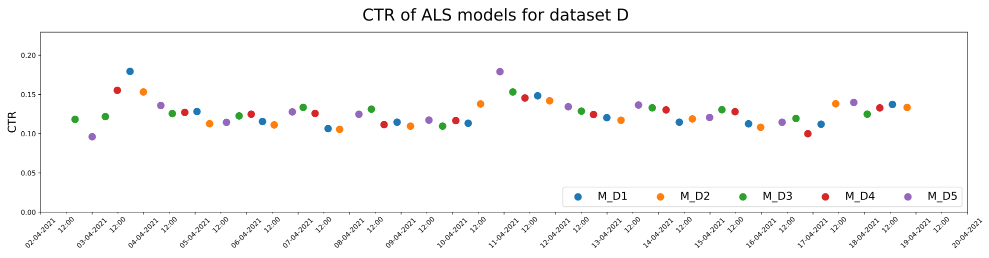


Figure 4.4: Changes of CTR over time for dataset D

Figure 4.5 shows no significant trend in the development of CTR for the dataset E representing the video streaming platform. However, the number of measurements for this dataset was not as large, and therefore, the daily fluctuations of CTR caused by the viewing of TV in the evening may not be captured on the plot.

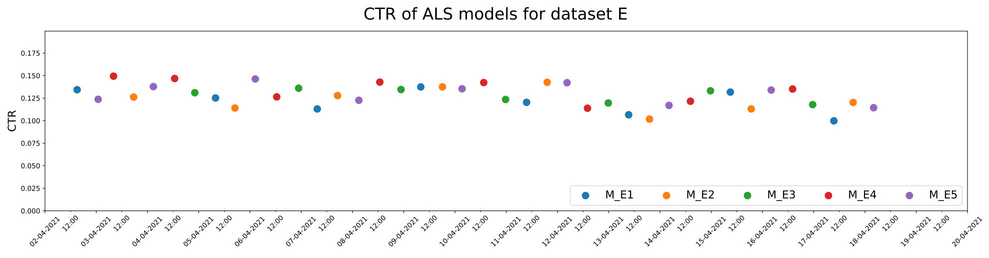


Figure 4.5: Changes of CTR over time for dataset E

The exact reasons for changes and fluctuations in the CTR can only be guessed. There are a number of reasons why changes may have occurred, ranging from social, economical to technical. The experiment cannot be reproduced with a goal to get identical results. This is one of the fundamental characteristics of CTR and a major problem for theoretical research. However, it is good to keep these changes in mind. That is why A/B tests usually take a number of weeks. There must be enough interaction and in a wide variety of circumstances to be able to tell which model is better. Why the CTR changes is not so important for this work, but it is important that it changes. This represents a big difference with recall that will be analysed now.

4.4 The effect of hyperparameters on recall

A new method of measuring recall called *popularity-stratified user-normalised leave(-last)-one-out recall* was introduced in (2.1), and (2.2), and contains three hyperparameters: β, VAL, K. Since there are no other parameters in some models, these hyperparameters will be called parameters in this section.

It will now be examined how these parameters affect the value of recall. During the experiment, 588 different parametrizations of recall for each model were measured, and each model was measured several times depending on how quickly the model changed. Therefore, the resulting dataframe contains 289,884 rows and cannot be presented here in full, but it is attached to this thesis.

First, the development of recall over time for the model with the highest number of factors (which should be most susceptible to overfitting) for each dataset was analyzed. It is important to remember that recall was repeatedly measured on the same set of test users who were separated before the experiment began. This is a big difference from CTR, which is measured largely on new users. For the experiment, the possibility that the test set of users will change during the experiment was also considered. However, this option would not reflect the demand for an offline evaluation metric that works with static data. Even if a relation between recall and CTRs is found, it would not be relevant for subsequent evaluation measured by recall.

As can be seen in Figure 4.6, recall remains more or less constant. The model changes between iterations but not enough to significantly change recall. The popularity of items is changing, both for the model and for the measurement of recall. Although recall in Figure 4.6 was measured with $\beta = 1$ and popularity is heavily penalised, changes of popularity do not have a significant effect in the short term when measuring recall. The popularity of items does not change so much because if the dataset has 30m interactions and there are 10k interactions per day, then the average popularity of an item changes by 0.3% per day. To monitor the effect of changes in popularity over time on recall, it would be useful to calculate popularity only from more recent interactions. This option was not explored in the experiment because the experiment focuses on using popularity penalisation for debiasing the data, and

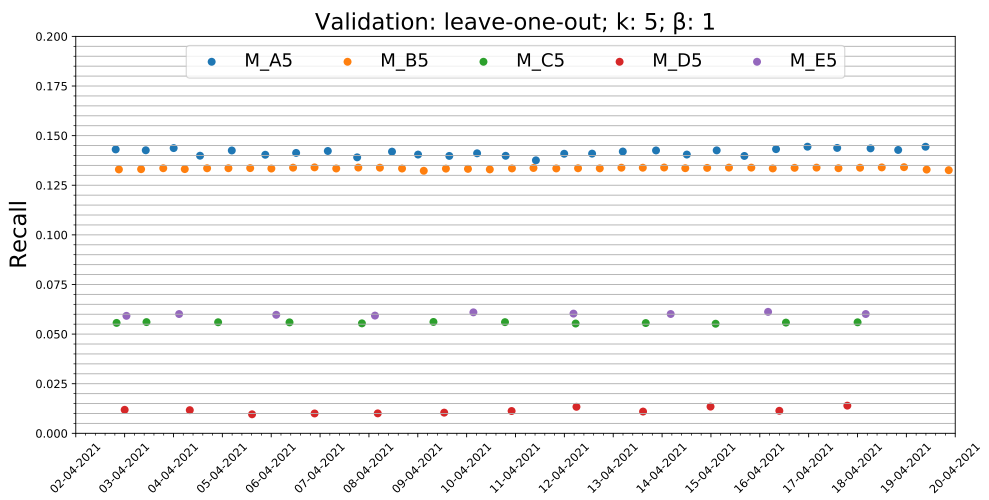


Figure 4.6: The development of recall over time for models

for that, it is better to calculate the overall popularity.

Another parameter that will be studied is the method of validation. Average recall for $K = 5$ and $\beta = 1$ for both validations tested can be seen in Table 4.3. It can be seen that *leave-one-out* recall is almost always greater than *leave-last-ne-out* recall. That is because of how the evaluated algorithm works. The whole experiment of based on recommendations created by the item-KNN algorithm, which is a nonsequential algorithm. On the input of the algorithm is a list of items interacted by the user, and similar items are recommended. The order of interactions is not taken into account in the item-KNN algorithm as opposed to algorithms such as the LSTM recommendation described in Thesis [2]. Item-KNN validated with LOO always uses at least that many interactions as LLOO for creating a recommendation, and thus makes sense that recall is greater.

Similar results can be found for any β and K values in the collected data. Comparing the average LOO recall of the model with the average LLOO recall, regardless of other parameters, it can be found that in $6081/7350 = 82.73\%$ of cases the LOO recall is higher than the LLOO recall. This number, equal to about 1/5, may indicate that one of the datasets is different from the others and that the LLOO recall is greater than the LOO recall for that dataset. And indeed it is, namely, the dataset D, as can be seen in Table 4.4. The cause of this phenomenon is unknown, but it may be related to the fact that dataset D is very specific. It contains only about 1k items, 70k users but 13.9m interactions. Therefore, it has a very dense rating matrix compared to other datasets.

The effect of the β parameter on recall will now be investigated. For the analysis, the results of measured recall were also analysed separately according

Model	average leave-one-out recall	average leave-last-one-out recall
M_{A1}	11.93%	6.53%
M_{A2}	15.33%	10.0%
M_{A3}	16.82%	12.62%
M_{A4}	16.07%	12.22%
M_{A5}	14.17%	11.36%
M_{B1}	6.71%	3.56%
M_{B2}	10.64%	5.63%
M_{B3}	11.59%	6.26%
M_{B4}	13.25%	7.98%
M_{B5}	13.35%	7.92%
M_{C1}	2.77%	1.96%
M_{C2}	3.59%	2.52%
M_{C3}	3.79%	2.71%
M_{C4}	4.8%	3.66%
M_{C5}	5.57%	4.22%
M_{D1}	3.89%	3.03%
M_{D2}	4.18%	4.08%
M_{D3}	4.92%	4.96%
M_{D4}	9.12%	6.65%
M_{D5}	1.15%	2.89%
M_{E1}	2.59%	1.9%
M_{E2}	2.81%	2.32%
M_{E3}	4.21%	3.72%
M_{E4}	5.64%	4.96%
M_{E5}	6.01%	5.08%

Table 4.3: Comparison of LOO and LLOO recall for $K = 5$ and $\beta = 1$

	Dataset A	Dataset B	Dataset C	Dataset D	Dataset E
#Samples	1470	1470	1470	210	1461
Portion	100%	100%	100%	14.29%	99.39%

Table 4.4: Portion of models and parameters aggregated for each dataset where LOO > LLOO recall

to the method of validation to capture any difference between penalisation with the LOO and LLOO validation.

Figure 4.8 shows the relation between recall and the β parameter for individual values of the K parameter. Moreover, the plot is divided into two columns according to the validation method. In the left column, the recall values are measured using *leave-one-out* validation, while the recall values measured using *leave-last-one-out validation* are located in the right column. Each M_{A_j} model for $j \in \{1, 2, 3, 4, 5\}$ is plotted in a separate row. Values of recall in the figure are calculated as the average values of recall for a given parametrization over time. The analogous description also applies to Figures 4.8, 4.9, 4.10 and 4.11 showing the same analysis, but for other datasets. The objective of the analysis is to examine how recall relates to β and if this relation depends on a domain, on a validation method, or on the K parameter.

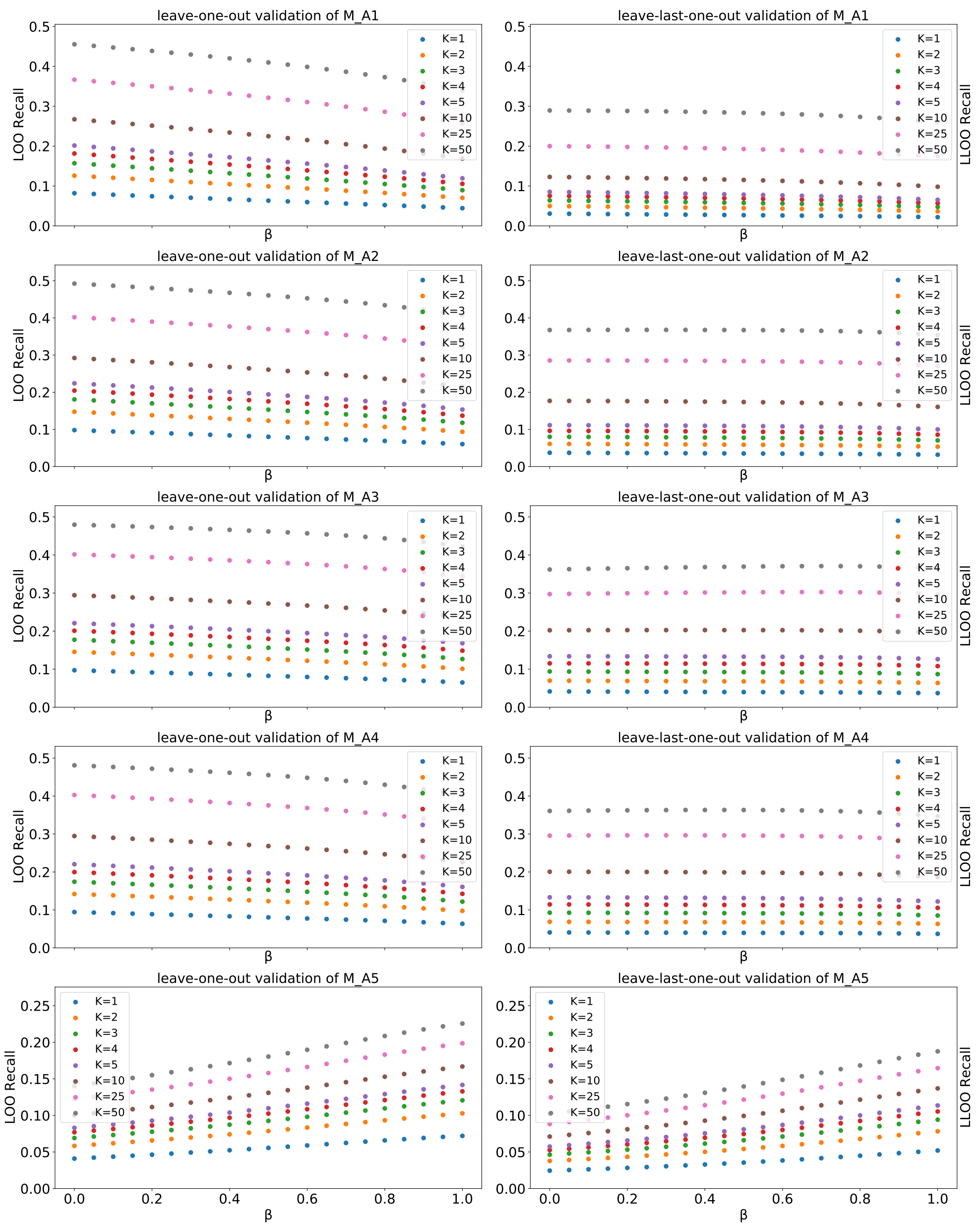


Figure 4.7: Comparison of β, validation method and recall for dataset A

Analysis in Figure 4.7 shows that neither the validation method nor the K parameter is important for the direction of the curve between β and recall. However, it appears that when measured using *leave-one-out* validation (left column), recall is more influenced by β than in the case of *leave-last-one-out* validation (right column), where a change in β does not affect recall that much. It is not clear if there is any dependency between recall and β across models. For the M_{A1} model, a negative correlation is seen between the β and recall, which means the bigger β results in the smaller recall. However, in the case of the M_{A5}, the trend is exactly the opposite, and there is a positive correlation between β and recall. It should be noted that at the beginning of the experiment, the models were ordered according to the number of factors that indicate computing capacity but also the susceptibility to overfitting. It means that the M_{A1} model, consisting of latent vectors with dimension 32, should be forced to generalize far more than the M_{A5} model, containing 1024-dimensional embeddings. Therefore, based on dataset A, it can be hypothesized that the effect of the β penalisation parameter is related to the computational capacity of the model.

Figure 4.8 displaying the analysis of the dataset B agrees with the statement that β has a greater impact on recall measured using the LOO method. However, the hypothesis of the relation between β and the number of factors is neither supported nor refuted. For all five models and both methods of validation, the bigger the β, the less recall. The model M_{B5} recommends on the basis of 1024-dimensional embeddings, but there is no repetition of the situation that increasing β increases recall. This may be due to the much larger dimension of the rating matrix that has been factorized into latent vectors with 1024 dimensions. Dataset A has about five times fewer users and seven times fewer items, so the rating matrix is much smaller than the rating matrix for dataset B, and the information compression to 1024-dimensional space can more easily end overfitted. No other dependencies or unexpected relations between parameters occurred on dataset B.

The plot of parameters (Figure 4.9) for dataset C more or less replicates the Figure 4.8 for dataset B and yields nothing new. Again, for all five models, there is a negative correlation between β and recall.

However, dataset D brings big changes. The first observation that has already been made before in Table 4.4 is now visually confirmed. LLOO recall is higher than the LOO recall for dataset D. There is also seen how much recall increases with the increasing K parameter. This is because the dataset has very few items, and it is therefore easy to recommend the relevant item for a growing K. What is very interesting is the effect of β on recall. For four of the five models, there is a negative correlation between β and recall, which has also been observed on other datasets. The M_{D4} model is an exception, as the LOO recall shows a positive correlation with β, and the LLOO recall first declines with increasing β, but then reverses and begins to slowly grow. Although, the M_{D4} model does not differ significantly in its parameters from others and the reason for this phenomenon is unknown.

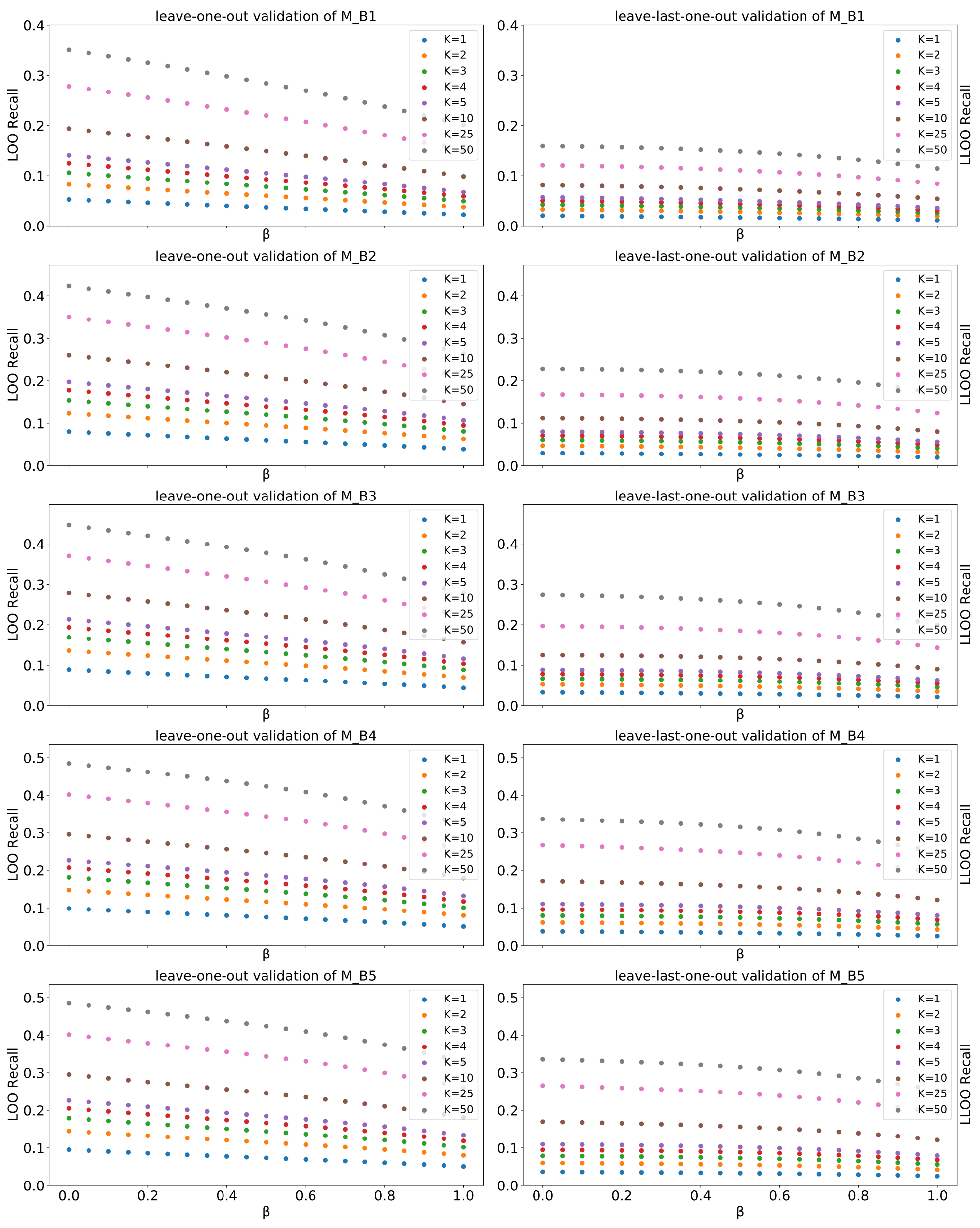


Figure 4.8: Comparison of β, validation method and recall for dataset B

The results of the dataset E shown in Figure 4.11 are uninteresting compared to dataset D. They are similar to datasets B and C. Although dataset E has a similar number of items and used as dataset A, there is no phenomenon that there is a positive correlation between β and recall for a 1024-dimensional model. Therefore, the hypothesis that increasing β could serve as a regularization to prevent overfitting seems to be false.

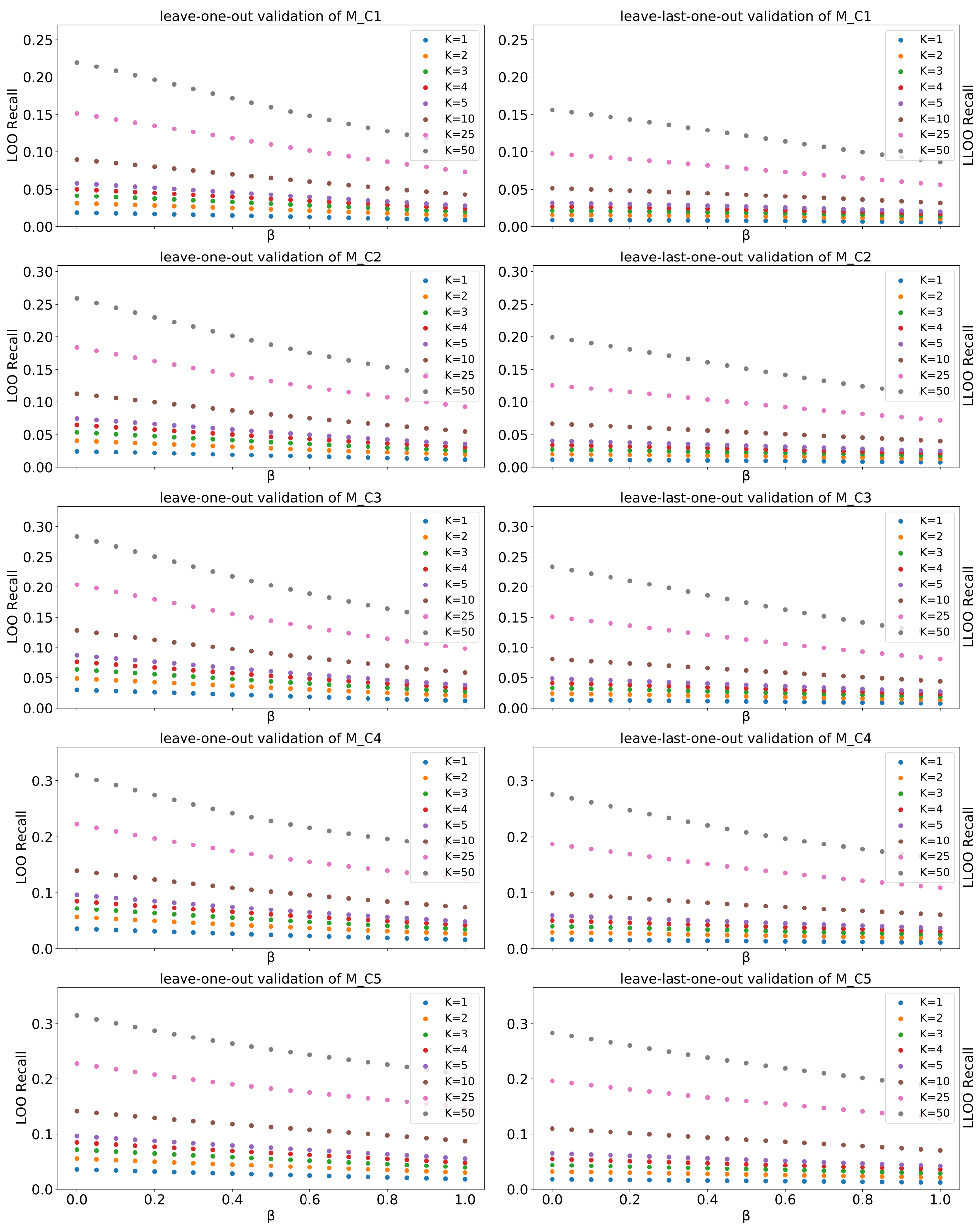


Figure 4.9: Comparison of β, validation method and recall for dataset C

The effect of the β parameter on PS-UN-LOO and PS-UN-LLOO recall was analyzed in this section. It has been experimentally verified that for most datasets and models, it is advisable to use LOO validation and not penalise popular items (β=0) to maximise recall. It will now be investigated how these hypeparametrizations relate to the online metric CTR and if higher recall implies higher CTR.

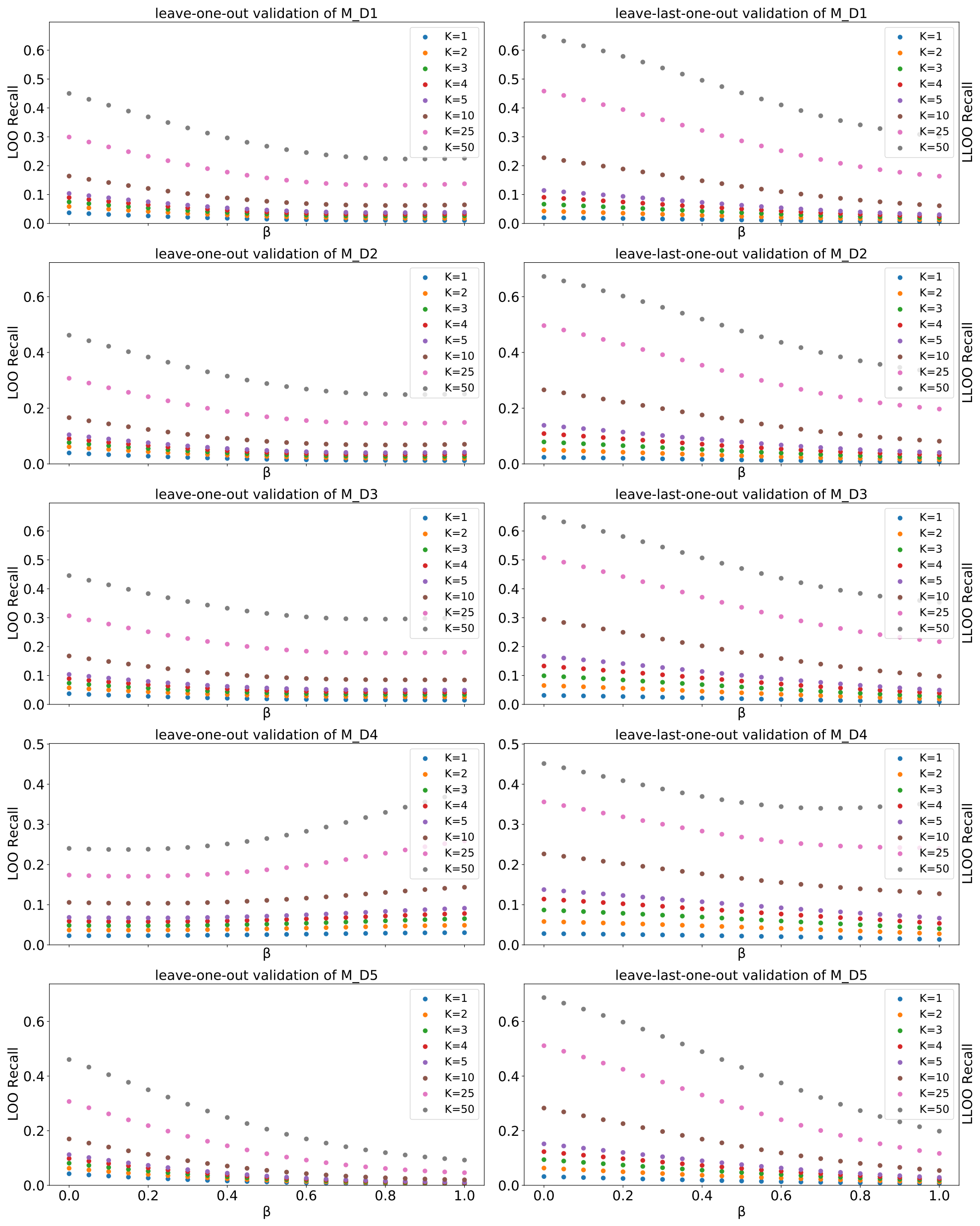


Figure 4.10: Comparison of β, validation method and recall for dataset D

4.5 Recall vs CTR

As described in Section 2.3, the CTR was aggregated by (2.7) to obtain an average value of CTR for each model. Similarly, recall was averaged over time by (2.6) and one value of recall was gained for each model and each parameterization.

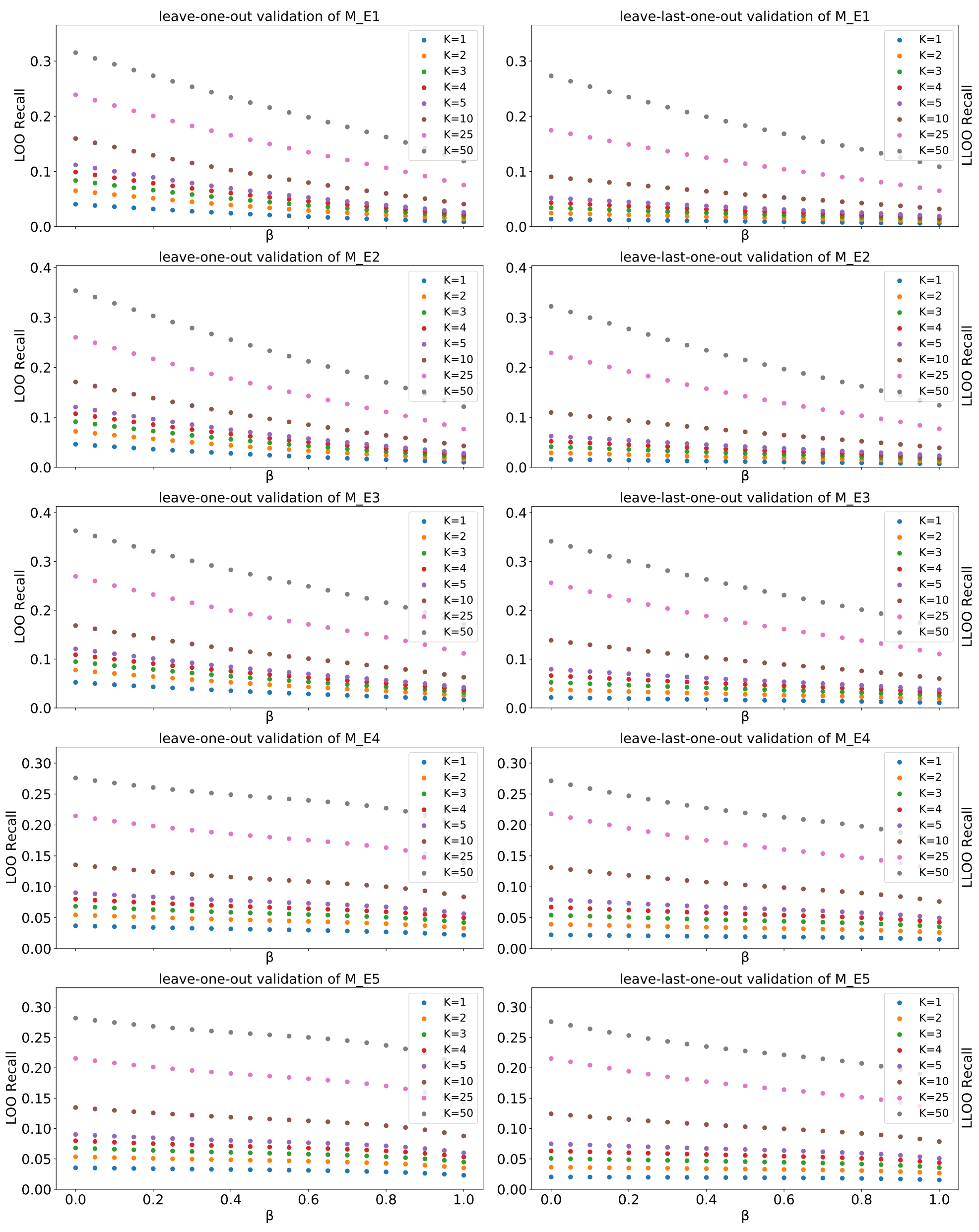


Figure 4.11: Comparison of β, validation method and recall for dataset E

A Spearman correlation coefficient was chosen to describe the correlation between CTR and recall. The commonly used Pearson correlation coefficient was not used since it does not recognize other than linear correlation. The aim of the experiment was to verify that the CTR and recall are monotonic, and thus the maximization of recall corresponds to the maximization of the CTR. The Spearman correlation coefficient (also called Rank coefficient) is more appropriate for this case because it does not describe the correlation between actual values but only between the order of values. The correlation was calculated between the average CTR and the average value of recall (for each parametrization) across the models of each dataset.

First, it was investigated whether there was any parameterization for which recall correlated with CTR for all datasets. The minimal, average, and maximum correlations across the datasets for selected parameterizations can be found in Table 4.5.

			Correlation		
β	*VAL*	K	minimal	average	maximum
0.30	leave-last-one-out	2	-0.4	0.34	0.9
0.35	leave-last-one-out	4	-0.4	0.34	0.9
0.90	leave-last-one-out	50	-0.4	0.34	0.8
0.15	leave-last-one-out	2	-0.4	0.34	0.9
0.15	leave-last-one-out	3	-0.4	0.34	0.9
0.15	leave-last-one-out	4	-0.4	0.34	0.9
0.20	leave-last-one-out	1	-0.4	0.34	0.9
0.20	leave-last-one-out	2	-0.4	0.34	0.9
0.20	leave-last-one-out	3	-0.4	0.34	0.9
0.20	leave-last-one-out	4	-0.4	0.34	0.9

Table 4.5: Minimal, average and maximum correlation for selected parametrizations of recall

Shown parametrizations are sorted by average correlation. At first glance, one can see that there is no parametrization that correlates for all datasets. For all parameter values in Table 4.5, recall significantly correlates with the CTR for one or more datasets, but not for all datasets. Parametrization of recall that correlates well with the CTR for one dataset may no longer correlate with the CTR of a second dataset. This finding implies the fundamental shortcomings of recall in the evaluation of RA and that maximising of recall does not always imply maximising CTR. This claim will now be investigated, and it will be determined on which dataset this does not apply. Another observation based on the Table 4.5 will be discussed later.

Correlation	[-1.0, -0.5]	(-0.5, 0.0]	(0.0, 0.5]	(0.5, 1.0]
Dataset A	7 (1.19%)	37 (6.29%)	451 (76.7%)	93 (15.82%)
Dataset B	0 (0%)	588 (100%)	0 (0%)	0 (0%)
Dataset C	0 (0%)	588 (100%)	0 (0%)	0 (0%)
Dataset D	9 (1.53%)	61 (10.37%)	334 (56.8%)	184 (31.29%)
Dataset E	69 (11.73%)	39 (6.63%)	58 (9.86%)	422 (71.77%)

Table 4.6: Comparison of correlation for each dataset

Table 4.6 shows that there are datasets for which recall does not correlate with CTR at all or even correlates negatively. For datasets B and C, maximization of recall could lead to a smaller CTR. The remaining datasets show a mostly positive correlation between recall and CTR. For all three remaining datasets, there are recall parameterizations, which correlate significantly (in (0.5 - 1]) with CTR. It will now be examined what those parametrizations are and if there is any relation between them across the datasets.

Table 4.7 plots the same information as Table 4.5, but is only based on datasets A, D, and E. Datasets B and C were temporarily omitted. Both average and minimum correlations can be seen to increase significantly.

			Correlation		
β	*VAL*	K	minimal	average	maximum
0.20	leave-last-one-out	2	0.5	0.77	0.9
0.25	leave-last-one-out	7	0.5	0.77	0.9
0.35	leave-last-one-out	3	0.5	0.77	0.9
0.15	leave-last-one-out	2	0.5	0.77	0.9
0.15	leave-last-one-out	3	0.5	0.77	0.9
0.15	leave-last-one-out	4	0.5	0.77	0.9
0.35	leave-last-one-out	4	0.5	0.77	0.9
0.35	leave-last-one-out	5	0.5	0.77	0.9
0.30	leave-last-one-out	5	0.5	0.77	0.9
0.30	leave-last-one-out	4	0.5	0.77	0.9

Table 4.7: Minimal, average and maximum correlation for selected parametrizations of recall for datasets A, D, E

For both Tables 4.5 and 4.7, the parameters with the highest average correlation include $VAL = LLOO$ and $\beta > 0$. Although not seen due to a large number of parametrizations, the highest average correlation of 0.77 occurs in 25 parametrizations and for all of them is the validation set to LLOO, and β is in the range $[0.15 - 0.4]$. Therefore, the LLOO validation appears to be a more appropriate method of validation than the LOO. Penalisation of popularity also seems to be successful.

A closer examination of these dependencies can be seen in Table 4.8 that shows the number of hyperparameterizations for which the correlation coefficient of the LOO validation was greater, equal to, or less than the correlation coefficient of the LLOO validation. The table also shows that the choice of validation method depends on a specific dataset. An extreme case is dataset C, which holds that the correlation coefficient between recall and CTR is virtually independent of the validation method, as it is the same for both methods in 99.32% of cases. A second interesting example is dataset D, for which LLOO recall is far better for CTR prediction. The opposite is dataset A, for which the LLOO recall is better than the LOO recall only in less than 5 per cent

	LOO > LLOO	LOO = LLOO	LOO < LLOO
Dataset A	150 (51.02%)	130 (44.22%)	14 (4.76%)
Dataset B	51 (17.35%)	243 (82.65%)	0 (0%)
Dataset C	2 (0.68%)	292 (99.32%)	0 (0%)
Dataset D	32 (10.88%)	47 (15.99%)	215 (73.13%)
Dataset E	9 (3.06%)	144 (48.98%)	141 (47.96%)
Dataset A + D + E	191 (21.66%)	321 (36.39%)	370 (41.95%)
Total	244 (16.6%)	856 (58.2%)	370 (25.2%)

Table 4.8: Comparison of correlations for different validation methods

of parametrizations. For a new unknown dataset, it would be recommended to validate it using the LLOO method, as in 83.04% of cases, the correlation between recall and CTR will be greater or equal than for LOO validation.

	$\beta_{=0} > \beta_{>0}$	$\beta_{=0} = \beta_{>0}$	$\beta_{=0} < \beta_{>0}$
Dataset A	248 (44.29%)	178 (31.79%)	134 (24.93%)
Dataset B	0 (0%)	475 (84.82%)	85 (15.18%)
Dataset C	40 (7.14%)	520 (92.86%)	0 (0%)
Dataset D	61 (10.89%)	93 (16.61%)	406 (72.5%)
Dataset E	23 (4.11%)	110 (19.64%)	427 (76.25%)
Dataset A + D + E	332 (19.76%)	381 (22.68%)	967 (57.56%)
Total	372 (13.29%)	1376 (49.14%)	1052 (37.57%)

Table 4.9: Comparison of correlations for popularity penalisation

Table 4.9 shows the effect of penalisation of popular items. Columns contain a number of parametrizations for

1. which no penalisation ($\beta = 0$) is better than some penalisation ($\beta > 0$),
2. which no penalisation and some penalisation have no effect on correlation,
3. which some penalisation is better than no penalisation.

Cases when parameterization with $\beta = 0$ was better for maximising the correlation coefficient between CTR and recall, are only 13.29% overall. However, as can be seen, the setting of β is domain-dependent because it is not recommended to penalise items in dataset A. However, in case of nothing is known about the dataset, it is recommended to use a penalisation.

The correlation between penalisation, validation, and correlation can be seen in Table 4.10, which contains the percentage of parametrizations (combination of K and dataset) according to the criteria specified in columns and rows. It can be seen that for the largest proportion of parametrisations (38%)

	LOO > LLOO	LOO = LLOO	LOO < LLOO	Total
$\beta_{=0} > \beta_{>0}$	6.69%	4.49%	2.11%	13.29%
$\beta_{=0} = \beta_{>0}$	6.04%	38.28%	4.83%	49.15%
$\beta_{=0} < \beta_{>0}$	3.87%	15.46%	18.24%	37.57%
Total	16.6%	58.23%	25.18%	100%

Table 4.10: The relation between penalisation, validation and corelation across all datasets

there is no relation between correlation, validation method and penalisation. However, the second-largest portion already shows that the correlation of CTR and recall is greater for LLOO and $\beta > 0$ combinations at 18%. This value may be increased. If datasets B and C are excluded from the analysis, then the use of penalisation and LLOO validation is appropriate to increase the correlation of recall with CTR, as shown in Table 4.11. Excluding dataset B, C simulates the situation that it is known that recall correlates at least a little with CTR, and the aim is to increase the correlation. In the case of a totally unknown dataset, when it is not known, what is the relation between recall and the CTR, it is also recommended to use LLOO and penalisation. This parameterization does not worsen the correlation between CTR and recall with probability 77%, and with probability 38.53% improves it.

	LOO > LLOO	LOO = LLOO	LOO < LLOO	Total
$\beta_{=0} > \beta_{>0}$	11.03%	5.22%	3.51%	19.76%
$\beta_{=0} = \beta_{>0}$	5.19%	9.44%	8.04%	22.67%
$\beta_{=0} < \beta_{>0}$	5.43%	21.73%	30.4%	57.56%
Total	21.65%	36.39%	41.95%	100%

Table 4.11: The relation between penalisation, validation and corelation across datasets A, D, E

It should be noted here that the PS-UN-LLOO recall presented in this work is precisely what seems to be best for finding the CTR-recall correlation. It beats previously published versions of recalls such as the popularity-stratified recall presented in article [61] or the LLOO recall described in Thesis [2]. PS-UN-LLOO combines both these approaches with additional user normalization as described in Thesis [33], and is experimentally verified to be appropriate for an unknown dataset as well as for a dataset with confirmed CTR-recall correlation where there is an intention to maximise the correlation. The analysis was based on data collected during the experiment. All this data is attached to the work for validation of the analysis as well as for analysing other relations. For example, in the experiment, catalog coverage was measured in addition to recall, so one can see the method of validation affects the overall catalog coverage.

4.6 Summary of results

During the chapter, the development of recall and CTR over time was shown. The finding is that while recall remains more or less constant over time, the CTR is very fluctuant. It was found that for some datasets, CTR oscillated periodically. The effect of hyperparameters on recall over time was subsequently analysed. It has been shown that the LOO validation achieves higher values of recall than LLOO validation. The cause of this phenomenon has also been explained for the item-KNN algorithm. Subsequently, it was shown for each dataset separately that for most models, recall decreases with increasing penalisation of popularity defined by β. Once the relation of hyperparameters and recall was analyzed, the impact of hyperparameters on the correlation of recall and CTR was investigated. It has been proven that greater recall does not always imply a greater CTR. Specifically, for two of the five datasets, the correlation (measured using a Spearman correlation coefficient) was between -0.5 and 0, i.e., the greater recall (regardless of hyperparameters) corresponded to the same or lower CTR. A slight positive correlation was shown for the remaining datasets. This correlation can be increased by correctly setting the hyperparameters of the recalls. In particular, the use of leave-last-one-out cross-validation and the penalisation of popular items by the β parameter has been found to improve the correlation between recall and CTR.

Chapter 5

Discussion

In industrial RS, the quality of the RA is measured using CTR as it is mostly correlated with business requirements. However, CTR metric is hard to measure without a live RS, so hundreds to thousands of researchers measure the quality of their RA using recall. It is widely assumed that there is a correlation between recall and CTR, and maximizing recall implies maximizing CTR. However, the thesis has shown that it is not always true. Recall is not an ideal metric for evaluating RAs. There are versions of recall described in this thesis that seem more suitable for evaluating RAs but still not good enough. The challenge for researchers remains to find a better metric that will be more relevant for industrial RSs, but with the possibility of repeatable evaluation without the need for real users. For the new evaluation method, it will be essential to correctly select test users to suppress the bias contained in most of the datasets. Another fundamental problem remains the question: *"What would the user do if he would receive another recommendation?"*. Understanding users' motivation and analysis of users' choices is far more important than blindly optimizing models to recall using increasingly computationally expensive models.

Conclusion

The work described the basic principles of Recommender Systems, with an emphasis on the evaluation of recommendation algorithms. It has been explained what interactions the Recommender System collects and how can RS recommend based on interactions. A recommendation algorithm based on matrix factorisation and an item-KNN algorithm has been described. Problems with interactions such as the MNAR problem and its implications for recommendation algorithms and evaluation have been pointed out. Two approaches of evaluation of recommendation algorithms were described in detail: offline and online. Both of them were compared theoretically and in practice. A framework was implemented capable of finding quality but different models for recommending items to real users. The models created were then recalculated periodically and evaluated using the implemented framework. In the implementation, emphasis was placed on the optimised representation of interactions in memory, as well as the efficient measurement of recall to handle large non-academic datasets. Thanks to this implementation, an extensive experiment was conducted to research the relation between recall and CTR.

The data collected in the experiment and the analysis performed confirmed that recall does not always correlate with CTR and is not suitable for optimizing recommendation algorithms used in industrial Recommender Systems. It was shown that on two out of five datasets, recall is not correlated at all with the CTR and on the other three datasets, it is only partially correlated. A version of recall called *popularity-stratified user-normalised leave-last-one-out recall* was presented. According to the results of the experiment, the proposed version of recall increases the correlation between recall and CTR for some datasets. However, even this modification could not reverse the proven fact that recall is not a good approximation of CTR. Finding a better metric remains a key problem for the research of Recommender Systems because it is not possible to systematically create quality recommendation algorithms without quality metrics.

Bibliography

[1] Kasalický, P. Content-Based Recommendation Model Trained Using Interaction Similarity. Accepted: 2018-06-19T21:57:26Z Publisher: České vysoké učení technické v Praze. Vypočetní a informační centrum. Available from: `https://dspace.cvut.cz/handle/10467/76817`

[2] Martínek, L. Doporučovací modely založené na rekurentních neuronových sítích. Accepted: 2020-06-14T10:41:02Z Publisher: České vysoké učení technické v Praze. Vypočetní a informační centrum. Available from: `https://dspace.cvut.cz/handle/10467/87990`

[3] Bhatnagar, V. *Collaborative Filtering Using Data Mining and Analysis*. IGI Global, ISBN 978-1-5225-0489-4.

[4] Schafer, J. B.; Frankowski, D.; et al. Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization*, Springer-Verlag, ISBN 978-3-540-72078-2, pp. 291–324.

[5] Herlocker, J. L.; Konstan, J. A.; et al. Evaluating collaborative filtering recommender systems. volume 22, no. 1: pp. 5–53, ISSN 1046-8188, doi:10.1145/963770.963772. Available from: `https://doi.org/10.1145/963770.963772`

[6] Schafer, J. B.; Konstan, J.; et al. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, EC '99, Association for Computing Machinery, ISBN 978-1-58113-176-5, pp. 158–166, doi:10.1145/336992.337035. Available from: `https://doi.org/10.1145/336992.337035`

[7] Karimi, M.; Jannach, D.; et al. News recommender systems – Survey and roads ahead. volume 54, no. 6: pp. 1203–1227, ISSN 0306-4573, doi:10.1016/j.ipm.2018.04.008. Available from: `https://www.sciencedirect.com/science/article/pii/S030645731730153X`

[8] Xu, Y.; Chen, Z.; et al. Learning to Recommend with User Generated Content. In *Web-Age Information Management*, edited by X. L. Dong; X. Yu; J. Li; Y. Sun, Lecture Notes in Computer Science, Springer International Publishing, ISBN 978-3-319-21042-1, pp. 221–232, doi: 10.1007/978-3-319-21042-1_18.

[9] Liu, S.; Wu, Q.; et al. Personalized Recommendation Considering Secondary Implicit Feedback. In *2018 IEEE International Conference on Agents (ICA)*, pp. 87–92, doi:10.1109/AGENTS.2018.8460053.

[10] Núñez-Valdéz, E. R.; Cueva Lovelle, J. M.; et al. Implicit feedback techniques on recommender systems applied to electronic books. volume 28, no. 4: pp. 1186–1193, ISSN 0747-5632, doi:10.1016/j.chb.2012.02.001. Available from: `https://www.sciencedirect.com/science/article/pii/S0747563212000325`

[11] Jawaheer, G.; Szomszor, M.; et al. Comparison of implicit and explicit feedback from an online music recommendation service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, Association for Computing Machinery, ISBN 978-1-4503-0407-8, pp. 47–51, doi: 10.1145/1869446.1869453. Available from: `https://doi.org/10.1145/1869446.1869453`

[12] Claypool, M.; Le, P.; et al. Implicit interest indicators. In *Proceedings of the 6th international conference on Intelligent user interfaces*, IUI '01, Association for Computing Machinery, ISBN 978-1-58113-325-7, pp. 33–40, doi:10.1145/359784.359836. Available from: `https://doi.org/10.1145/359784.359836`

[13] Koren, Y.; Bell, R. Advances in Collaborative Filtering. In *Recommender Systems Handbook*, edited by F. Ricci; L. Rokach; B. Shapira; P. B. Kantor, Springer US, ISBN 978-0-387-85820-3, pp. 145–186, doi:10.1007/978-0-387-85820-3_5. Available from: `https://doi.org/10.1007/978-0-387-85820-3_5`

[14] Oard, D. W.; Kim, J. Implicit Feedback for Recommender Systems: p. 3. Available from: `https://www.aaai.org/Papers/Workshops/1998/WS-98-08/WS98-08-021.pdf`

[15] Mobasher, B.; Cooley, R.; et al. Automatic personalization based on Web usage mining. volume 43, no. 8: pp. 142–151, ISSN 0001-0782, doi:10.1145/345124.345169. Available from: `https://doi.org/10.1145/345124.345169`

[16] Ricci, F.; Rokach, L.; et al. Recommender Systems: Introduction and Challenges. In *Recommender Systems Handbook*, edited by F. Ricci;

L. Rokach; B. Shapira, Springer US, ISBN 978-1-4899-7637-6, pp. 1–34, doi:10.1007/978-1-4899-7637-6_1. Available from: `https://doi.org/10.1007/978-1-4899-7637-6_1`

[17] Hu, Y.; Koren, Y.; et al. Collaborative Filtering for Implicit Feedback Datasets. pp. 263–272, doi:10.1109/ICDM.2008.22.

[18] Marlin, B. M.; Zemel, R. S.; et al. Collaborative filtering and the missing at random assumption. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, UAI'07, AUAI Press, ISBN 978-0-9749039-3-4, pp. 267–275.

[19] Marlin, B. M.; Zemel, R. S. Collaborative prediction and ranking with non-random missing data. In *Proceedings of the third ACM conference on Recommender systems*, RecSys '09, Association for Computing Machinery, ISBN 978-1-60558-435-5, pp. 5–12, doi:10.1145/1639714.1639717. Available from: `https://doi.org/10.1145/1639714.1639717`

[20] Anand, S. S.; Kearney, P.; et al. Generating semantically enriched user profiles for Web personalization. volume 7, no. 4: pp. 22–es, ISSN 1533-5399, doi:10.1145/1278366.1278371. Available from: `https://doi.org/10.1145/1278366.1278371`

[21] Jawaheer, G.; Weller, P.; et al. Modeling User Preferences in Recommender Systems: A Classification Framework for Explicit and Implicit User Feedback. volume 4, no. 2: pp. 1–26, ISSN 2160-6455, 2160-6463, doi:10.1145/2512208. Available from: `https://dl.acm.org/doi/10.1145/2512208`

[22] Likert, R. A technique for the measurement of attitudes. volume 140: pp. 1–55.

[23] Amatriain, X.; Pujol, J. M.; et al. I Like It... I Like It Not: Evaluating User Ratings Noise in Recommender Systems. In *User Modeling, Adaptation, and Personalization*, edited by G.-J. Houben; G. McCalla; F. Pianesi; M. Zancanaro, Lecture Notes in Computer Science, Springer, ISBN 978-3-642-02247-0, pp. 247–258, doi:10.1007/978-3-642-02247-0_24.

[24] Gadanho, S. C.; Lhuillier, N. Addressing uncertainty in implicit preferences. In *Proceedings of the 2007 ACM conference on Recommender systems*, RecSys '07, Association for Computing Machinery, ISBN 978-1-59593-730-8, pp. 97–104, doi:10.1145/1297231.1297248. Available from: `https://doi.org/10.1145/1297231.1297248`

[25] Koprinska, I.; Yacef, K. People-to-People Reciprocal Recommenders. In *Recommender Systems Handbook*, edited by F. Ricci; L. Rokach; B. Shapira, Springer US, ISBN 978-1-4899-7637-6, pp. 545–567,

doi:10.1007/978-1-4899-7637-6_16. Available from: `https://doi.org/10.1007/978-1-4899-7637-6_16`

[26] Koren, Y.; Bell, R.; et al. Matrix Factorization Techniques for Recommender Systems. volume 42, no. 8: pp. 30–37, ISSN 1558-0814, doi:10.1109/MC.2009.263, conference Name: Computer.

[27] Goldberg, D.; Nichols, D.; et al. Using collaborative filtering to weave an information tapestry. volume 35, no. 12: pp. 61–70, ISSN 0001-0782, doi:10.1145/138859.138867. Available from: `https://doi.org/10.1145/138859.138867`

[28] Ning, X.; Desrosiers, C.; et al. A Comprehensive Survey of Neighborhood-Based Recommendation Methods. In *Recommender Systems Handbook*, edited by F. Ricci; L. Rokach; B. Shapira, Springer US, ISBN 978-1-4899-7637-6, pp. 37–76, doi:10.1007/978-1-4899-7637-6_2. Available from: `https://doi.org/10.1007/978-1-4899-7637-6_2`

[29] Collaborative Filtering. In *Encyclopedia of Machine Learning*, edited by C. Sammut; G. I. Webb, Springer US, ISBN 978-0-387-30164-8, pp. 189–189, doi:10.1007/978-0-387-30164-8_138. Available from: `https://doi.org/10.1007/978-0-387-30164-8_138`

[30] Hofmann, T. Latent semantic models for collaborative filtering. volume 22, no. 1: pp. 89–115, ISSN 1046-8188, doi:10.1145/963770.963774. Available from: `https://doi.org/10.1145/963770.963774`

[31] Karypis, G. Evaluation of Item-Based Top-N Recommendation Algorithms. In *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, Association for Computing Machinery, ISBN 978-1-58113-436-0, pp. 247–254, doi:10.1145/502585.502627. Available from: `https://doi.org/10.1145/502585.502627`

[32] Lee, T. Q.; Park, Y.; et al. A time-based approach to effective recommender systems using implicit feedback. volume 34, no. 4: pp. 3055–3062, ISSN 0957-4174, doi:10.1016/j.eswa.2007.06.031. Available from: `https://www.sciencedirect.com/science/article/pii/S0957417407002357`

[33] Řehořek, T. Manipulating the Capacity of Recommendation Models in Recall-Coverage Optimization. Accepted: 2019-04-05T11:19:10Z Publisher: České vysoké učení technické v Praze. Vypočetní a informační centrum. Available from: `https://dspace.cvut.cz/handle/10467/81823`

[34] Deshpande, M.; Karypis, G. Item-based top-N recommendation algorithms. volume 22, no. 1: pp. 143–177, ISSN 1046-8188, doi:10.1145/963770.963776. Available from: `https://doi.org/10.1145/963770.963776`

[35] Ostuni, V. C.; Di Noia, T.; et al. Top-N recommendations from implicit feedback leveraging linked open data. In *Proceedings of the 7th ACM conference on Recommender systems*, RecSys '13, Association for Computing Machinery, ISBN 978-1-4503-2409-0, pp. 85–92, doi: 10.1145/2507157.2507172. Available from: `https://doi.org/10.1145/2507157.2507172`

[36] Ren, Y.; Zhu, T.; et al. Top-N Recommendations by Learning User Preference Dynamics. In *Advances in Knowledge Discovery and Data Mining*, edited by J. Pei; V. S. Tseng; L. Cao; H. Motoda; G. Xu, Lecture Notes in Computer Science, Springer, ISBN 978-3-642-37456-2, pp. 390–401, doi:10.1007/978-3-642-37456-2_33.

[37] Celma, O. The Long Tail in Recommender Systems. In *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*, edited by O. Celma, Springer, ISBN 978-3-642-13287-2, pp. 87–107, doi:10.1007/978-3-642-13287-2_4. Available from: `https://doi.org/10.1007/978-3-642-13287-2_4`

[38] Latent Factor Models and Matrix Factorizations. In *Encyclopedia of Machine Learning*, edited by C. Sammut; G. I. Webb, Springer US, ISBN 978-0-387-30164-8, pp. 571–571, doi:10.1007/978-0-387-30164-8_887. Available from: `https://doi.org/10.1007/978-0-387-30164-8_887`

[39] Cheng, W.; Shen, Y.; et al. Explaining Latent Factor Models for Recommendation with Influence Functions: pp. 885–893. doi:10.1145/3292500.3330857, 1811.08120. Available from: `http://arxiv.org/abs/1811.08120`

[40] Bell, R. M.; Koren, Y. Lessons from the Netflix prize challenge. volume 9, no. 2: pp. 75–79, ISSN 1931-0145, doi:10.1145/1345448.1345465. Available from: `https://doi.org/10.1145/1345448.1345465`

[41] Amatriain, X.; Pujol, J. M. Data Mining Methods for Recommender Systems. In *Recommender Systems Handbook*, edited by F. Ricci; L. Rokach; B. Shapira, Springer US, ISBN 978-1-4899-7637-6, pp. 227–262, doi:10.1007/978-1-4899-7637-6_7. Available from: `https://doi.org/10.1007/978-1-4899-7637-6_7`

[42] Golub, G. H.; Reinsch, C. Singular Value Decomposition and Least Squares Solutions. In *Handbook for Automatic Computation: Volume II: Linear Algebra*, edited by J. H. Wilkinson; C. Reinsch; F. L. Bauer; A. S. Householder; F. W. J. Olver; H. Rutishauser; K. Samelson; E. Stiefel, Die Grundlehren der mathematischen Wissenschaften, Springer, ISBN 978-3-642-86940-2, pp. 134–151, doi:10.1007/978-3-642-86940-2_10. Available from: `https://doi.org/10.1007/978-3-642-86940-2_10`

[43] Holeňa, M.; Pulc, P.; et al. Important Internet Applications of Classification. In *Classification Methods for Internet Applications*, edited by M. Holeňa; P. Pulc; M. Kopp, Studies in Big Data, Springer International Publishing, ISBN 978-3-030-36962-0, pp. 1–68, doi:10.1007/978-3-030-36962-0_1. Available from: https://doi.org/10.1007/978-3-030-36962-0_1

[44] He, X.; Liao, L.; et al. Neural Collaborative Filtering. 1708.05031. Available from: http://arxiv.org/abs/1708.05031

[45] Gilotte, A.; Calauzènes, C.; et al. Offline A/B testing for Recommender Systems: pp. 198–206. doi:10.1145/3159652.3159687, 1801.07030. Available from: http://arxiv.org/abs/1801.07030

[46] Cañamares, R.; Castells, P.; et al. Offline evaluation options for recommender systems. volume 23, no. 4: pp. 387–410, ISSN 1573-7659, doi:10.1007/s10791-020-09371-3. Available from: https://doi.org/10.1007/s10791-020-09371-3

[47] Liu, F.; Tang, R.; et al. Deep Reinforcement Learning based Recommendation with Explicit User-Item Interactions Modeling. 1810.12027. Available from: http://arxiv.org/abs/1810.12027

[48] Koren, Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, Association for Computing Machinery, ISBN 978-1-60558-193-4, pp. 426–434, doi:10.1145/1401890.1401944. Available from: https://doi.org/10.1145/1401890.1401944

[49] Vinagre, J.; Jorge, A. M.; et al. Evaluation of recommender systems in streaming environments. doi:10.13140/2.1.4381.5367, 1504.08175. Available from: http://arxiv.org/abs/1504.08175

[50] Bennett, J.; Lanning, S.; et al. The Netflix Prize. In *In KDD Cup and Workshop in conjunction with KDD.*

[51] Dror, G.; Koenigstein, N.; et al. The Yahoo! Music Dataset and KDD-Cup'11. In *Proceedings of the 2011 International Conference on KDD Cup 2011 - Volume 18*, KDDCUP'11, JMLR.org, pp. 3–18.

[52] Goldberg, K.; Roeder, T.; et al. Eigentaste: A Constant Time Collaborative Filtering Algorithm. volume 4, no. 2: pp. 133–151, ISSN 1573-7659, doi:10.1023/A:1011419012209. Available from: https://doi.org/10.1023/A:1011419012209

[53] Järvelin, K.; Kekäläinen, J. Cumulated gain-based evaluation of IR techniques. volume 20, no. 4: pp. 422–446, ISSN 1046-8188, doi:10.1145/582415.582418. Available from: https://doi.org/10.1145/582415.582418

[54] Liang, D.; Krishnan, R. G.; et al. Variational Autoencoders for Collaborative Filtering. 1802.05814. Available from: http://arxiv.org/abs/1802.05814

[55] Zhu, Z.; Wang, J.; et al. Improving Top-K Recommendation via JointCollaborative Autoencoders. In *The World Wide Web Conference*, WWW '19, Association for Computing Machinery, ISBN 978-1-4503-6674-8, pp. 3483–3482, doi:10.1145/3308558.3313678. Available from: https://doi.org/10.1145/3308558.3313678

[56] Kim, D.; Suh, B. Enhancing VAEs for Collaborative Filtering: Flexible Priors & Gating Mechanisms. doi:10.1145/3298689.3347015, 1911.00936. Available from: http://arxiv.org/abs/1911.00936

[57] Campochiaro, E.; Casatta, R.; et al. Do Metrics Make Recommender Algorithms? In *2009 International Conference on Advanced Information Networking and Applications Workshops*, pp. 648–653, doi:10.1109/WAINA.2009.127.

[58] Shani, G.; Gunawardana, A. Evaluating Recommendation Systems. In *Recommender Systems Handbook*, edited by F. Ricci; L. Rokach; B. Shapira; P. B. Kantor, Springer US, ISBN 978-0-387-85820-3, pp. 257–297, doi:10.1007/978-0-387-85820-3_8. Available from: https://doi.org/10.1007/978-0-387-85820-3_8

[59] Rehorek, T.; Biza, O.; et al. Comparing Offline and Online Evaluation Results of Recommender Systems: p. 5.

[60] Carraro, D.; Bridge, D. Debiased offline evaluation of recommender systems: a weighted-sampling approach. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, Association for Computing Machinery, ISBN 978-1-4503-6866-7, pp. 1435–1442, doi:10.1145/3341105.3375759. Available from: https://doi.org/10.1145/3341105.3375759

[61] Steck, H. Item popularity and recommendation accuracy. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, Association for Computing Machinery, ISBN 978-1-4503-0683-6, pp. 125–132, doi:10.1145/2043932.2043957. Available from: https://doi.org/10.1145/2043932.2043957

[62] Li, L.; Kim, J. Y.; et al. Toward Predicting the Outcome of an A/B Experiment for Search Relevance. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, WSDM '15, Association for Computing Machinery, ISBN 978-1-4503-3317-7, pp. 37–46, doi:10.1145/2684822.2685311. Available from: https://doi.org/10.1145/2684822.2685311

[63] Beel, J.; Genzmehr, M.; et al. A comparative analysis of offline and online evaluations and discussion of research paper recommender system evaluation. In *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation*, RepSys '13, Association for Computing Machinery, ISBN 978-1-4503-2465-6, pp. 7–14, doi:10.1145/2532508.2532511. Available from: https://doi.org/10.1145/2532508.2532511

[64] Chagniot, P.; Vasile, F.; et al. From Clicks to Conversions: Recommendation for long-term reward. 2009.00497. Available from: http://arxiv.org/abs/2009.00497

[65] Zhou, R.; Khemmarat, S.; et al. The impact of YouTube recommendation system on video views. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, IMC '10, Association for Computing Machinery, ISBN 978-1-4503-0483-2, pp. 404–410, doi:10.1145/1879141.1879193. Available from: https://doi.org/10.1145/1879141.1879193

[66] Chen, L.; Pu, P. Eye-Tracking Study of User Behavior in Recommender Interfaces. In *User Modeling, Adaptation, and Personalization*, edited by P. De Bra; A. Kobsa; D. Chin, Lecture Notes in Computer Science, Springer, ISBN 978-3-642-13470-8, pp. 375–380, doi:10.1007/978-3-642-13470-8_35.

[67] Pu, P.; Chen, L.; et al. A user-centric evaluation framework for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, Association for Computing Machinery, ISBN 978-1-4503-0683-6, pp. 157–164, doi:10.1145/2043932.2043962. Available from: https://doi.org/10.1145/2043932.2043962

[68] Kohavi, R.; Longbotham, R.; et al. Controlled experiments on the web: survey and practical guide. volume 18, no. 1: pp. 140–181, ISSN 1573-756X, doi:10.1007/s10618-008-0114-1. Available from: https://doi.org/10.1007/s10618-008-0114-1

[69] Peska, L.; Vojtas, P. Off-line vs. On-line Evaluation of Recommender Systems in Small E-commerce: pp. 291–300. doi:10.1145/3372923.3404781, 1809.03186. Available from: http://arxiv.org/abs/1809.03186

[70] Amatriain, X.; Basilico, J. Recommender Systems in Industry: A Netflix Case Study. In *Recommender Systems Handbook*, edited by F. Ricci; L. Rokach; B. Shapira, Springer US, ISBN 978-1-4899-7637-6, pp. 385–419, doi:10.1007/978-1-4899-7637-6_11. Available from: `https://doi.org/10.1007/978-1-4899-7637-6_11`

[71] Garcin, F.; Faltings, B.; et al. Offline and online evaluation of news recommender systems at swissinfo.ch. In *Proceedings of the 8th ACM Conference on Recommender systems*, RecSys '14, Association for Computing Machinery, ISBN 978-1-4503-2668-1, pp. 169–176, doi: 10.1145/2645710.2645745. Available from: `https://doi.org/10.1145/2645710.2645745`

[72] Gama, J.; Sebastião, R.; et al. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, Association for Computing Machinery, ISBN 978-1-60558-495-9, pp. 329–338, doi: 10.1145/1557019.1557060. Available from: `https://doi.org/10.1145/1557019.1557060`

Appendix **A**

Acronyms

ALS	Alternating Least Squares
CC	Catalog coverage
CF	Collaborative Filtering
CTR	Click-through rate
iCTR	Implicit click-through rate
EF	Explicit Feedback
IF	Implicit Feedback
LFM	Latent factor models
LLOO	Leave-last-one-out
LOO	Leave-one-out
MF	Matrix factorisation
MNAR	Missing-not-at-random
MSE	Mean Square Error
NN	Nearest neighbours
PS	Popularity-stratified
RA	Recommendation Algorithm
RBE	Ranking-based evaluation
RMSE	Root Mean Square Error
RPBE	Rating prediction-based evaluation

RS	Recommender System
SVD	Singular Value Decomposition
UN	User-normalised

Appendix **B**

Contents of enclosed CD

```
src
 ├── implementation ........... source codes of the framework in Python
 └── thesis .. source codes of the thesis in LaTeX format along with images
results
 ├── thesis.pdf ................................ thesis in PDF format
 ├── recall.csv ... CSV file with 289.885 rows containing measured recall
 ├── ctr.csv .......... CSV file with 494 rows containing measured CTR
 └── ctr-recall-example.ipynb ... Example of work with recall and CTR
```