FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Manipulating the Capacity of Recommendation Models in Recall-Coverage Optimization

by

Tomáš Řehořek

A dissertation thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

Dissertation degree study programme: Informatics

Department of Applied Mathematics

Prague, August 2018

Supervisor:
doc. Pavel Kordík, Ph.D.
Department of Applied Mathematics
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Abstract and contributions

Traditional approaches in Recommender Systems ignore the problem of long-tail recommendations. There is no systematic approach to control the magnitude of long-tail recommendations generated by the models, and there is not even proper methodology to evaluate the quality of long-tail recommendations. This thesis addresses the long-tail recommendation problem from both the algorithmic and evaluation perspective. We proposed controlling the magnitude of long-tail recommendations generated by models through the manipulation with capacity hyperparameters of learning algorithms, and we define such hyperparameters for multiple state-of-the-art algorithms. We also summarize multiple such algorithms under the common framework of the *score* function, which allows us to apply popularity-based regularization to all of them. We propose searching for Pareto-optimal states in the Recall-Coverage plane as the right way to search for long-tail, high-accuracy models. On the set of exhaustive experiments, we empirically demonstrate the corectness of our theory on a mixture of public and industrial datasets for 5 different algorithms and their different versions.

In particular, the main contributions of the dissertation thesis are as follows:

- Unified framework for generating rating matrices from mixed data sources, including both the explicit and implicit ratings.
- Summarizing multiple state-of-the-art recommendation algorithms under the common framework based on the $score: I \times U \rightarrow \mathbb{R}$ function.
- Generalizing Collaborative Filtering ItemKnn algorithm as proposed in [93] to $sim: I^2 \rightarrow \mathbb{R}$ functions other than rating cosine similarity. The newly proposed similarity functions include the *tokenized attributes* similarity, *embedding* similarity, and *predicted* similarity. This allows using the algorithm for Content-Based recommendation as well.
- Association Rules for recommendation. While this is type of model didn't receive enough attention in the research community, we proposed unified framework for rule-

based recommendation, including the best-rule method, novel method of weighted voting, and different rule-quality measures. We also showed how the β-boosting (popularity regularization) applied to ARs translates to continous transition from *confidence* to *lift*. In the experiments, we showed that rule-based recommenders can compete to other state-of-the-art models.

- Survey of Matrix Factorization methods and Deep Learning approaches in Recommender Systems.
- Formal framework for hyperparameterizable learning algorithms producing recommendation models. Generalized definion of validation reward function, capturing not only accuracy measures, but other measures (inluding catalog and user coverage) as well. Unifying UserKnn, ItemKnn, ARs, MF, and AutoRec under this framework.
- Survey to Top-N evaluation in Recommender Systems.
- Generalization of popularity-based regularization, proposed in [101] for one specific MF algorithm, to all the algorithms presented in the State-of-the-Art chapter.
- Proposing model capacity manipulation as a method of controlling magnitude of long-tail recommendations.
- Defining model capacity manipulation parameters for UserKnn, ItemKnn, ARs, MF, and AutoRec.
- Proposing multi-objective top-N evaluation as searching for Pareto-optimal front in the hyperparameter space of learning algorithms.
- Proposing recall-coverage plane as highly relevant for optimizing long-tail recommendations.
- Exhaustive set of experiments on a mixture of 7 public and industrial datasets. Thorough investigation of manipulation with algorithm-dependent capacity parameters and the response in the recall-coverage plane.
- For UserKnn, showing that recall-coverage behavior is preserved even for Locality-Sensitive Hashing (LSH).

Keywords:
Recommender Systems, Long Tail, Hyperparameterization, Model Capacity, Regularization, Popularity, Collaborative Filtering, Recall, Catalog Coverage, Multi-Objective Optimization

Acknowledgements

There are many people to whom I would like express my gratitude for supporting me during my whole studies and also during writing this thesis.

I would like to thank my loving mother, Pavla Rehorkova, Ph.D, for her moral and material support during my whole studies. I would like to thank my loving grandparents, Frantisek Petrzela (†2015) and Emilie Petrzelova (†2017), for the very same, and for keeping asking me about the progress of my doctoral study until their very last days. It was Frantisek Petrzela's example what taught me to distinguish between seemingly impossible task under hostile conditions, and pure laziness. The conditions for finishing my thesis were actually wonderful and excellent, thanks to my loving wife and amazing woman, Anna Rehorkova, who let me leave her for several weeks just to write this thesis, and managed purchasing our new apartment, including mortgage negotiation, while I was gone. I wrote this thesis in beautiful summer residence, that my mother Pavla and her companion Michal Novak have been restoring for years and let me occupy it like a nobleman. I would also like to thank my father, Milan Rehorek, for supporting me throughout my whole studies as well.

This thesis is mostly theoretical, but when I sometimes put in a practical comment, it comes from Recombee, a company where we run Recommendation as a Service for dozens of international clients. I would like to thank my supervisor, Pavel Kordik, Ph.D., who gave the impetus to start the company, so we could get access to hundreds of real-world, industrial datasets, which is something that most researches don't have, making their research tied to few public datasets only. It may be connected with personality, but I believe that the neutral land on the interface between theory and practice is the most fertile. The things we are doing in Recombee were from the beginning highly influenced by my research, and my research have been continuously influenced by feedback from Recombee. Running production system differs from scientific experiments in that all 100 physical servers (as of 2018) must be up and running in 365/24/7 manner. I would therefore like to thank my colleagues who allowed me to leave the company for several weeks as well, without worries that anything will get wrong, because they are simply elite. They include Ondrej Fiedler (the best software engineer I've ever met), Ivan Povalyaev (ultimate hacker),

Gabriela Takacova (empathic business expert taking care of all the clients), Radek Bartyzal (researcher keeping pace with the latest trends), and Tomas Barton (infrastructure expert).

Furthermore, I would like to express my gratitude to many highly talented students who I had the opportunity to supervise on their Bachelor or Master theses, or those who I could cooperate with on their semestral projects. Without them, this thesis couldn't originate, because it was the long term cooperation with them which kept pushing my knowledge and the experimental results in Recommender Systems forward. From the students whose theses I supervised, I'd especially like to thank to Michal Bajer, Martin Barus, Tomas Fedor, Tomas Fryda, Martin Hak, Petr Kasalicky, Filip Krestan, Ladislav Martinek, Martin Pavlicek, Michal Reznicky, and Tomas Richtr. Many of them were deservedly awarded the Dean's Award, simply because they are excellent. From students I could cooperate on their semestral projects, I'd especially like to thank Ondrej Biza, Jakub Drdak, Josef Dvorak, Ondrej Novak, and Ondrej Podsztavek, some of whom shifted by knowledge really far. Sadly, the probability I forgot about someone is equal to 1, but again, thank you all for being so great, it was you who made my academic work so enjoyable.

Special thanks belongs also to industrial partners. I first got in touch with Recommender Systems in practice 7 years ago in nangu.TV (Alnair, a.s.), and continued in real-world delivery to VoD systems for Icflix and ShowMax companies since then, all thanks to Antonin Kral, Ph.D. The experimental part of this thesis is highly based on industrial datasets provided by Recombee clients, who kindly gave me their permission to perform experiments on their data. I would like to thank Nicholas Blicker Larsen from Design Group ApS. (Denmark) for providing data from Moodings and Just Spotted online stores, Giovanni Bartoli (Italy) for providing data from Casa Cenina store, and Lucas Colette from BUBB.Store (Brazil) for providing dataset covering multiple stores under their brilliant platform. Considering already mentioned Recombee, I would like to thank for providing expensive yet powerful hardware for my computationally-intensive experiments.

My research has also been supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS (SGS10/307/OHK3/3T/18, "Novel Model Ensembling Algorithms"), and

Dedication

To my wife Anna

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

Number Sets

$\mathbb{N}$	Natural numbers set
$\mathbb{Z}$	Integer numbers set
$\mathbb{R}$	Real numbers set
$\mathbb{R}_0^+$	Set of real numbers greater or equal to zero (that is $\{x \in R \mid x \geq 0\}$)
$\mathbb{R}^+$	Set of real numbers greater than zero (that is $\{x \in R \mid x > 0\}$)
$(\mathbb{R} \cup \{?\})$	Set of real numbers including unobserved (unknown) value

Common Mathematical Functions and Operators

$\mathbf{x}$	Column vector $\mathbf{x}$
x_i	i-th index of vector $\mathbf{X}$
$\|\|\mathbf{x}\|\|$	Frobenius norm of vector $\mathbf{b}$
$\mathbf{A}$	Matrix $\mathbf{A}$
$\mathbf{I}$	Identity matrix
$\mathbf{A}^T$	Transposed matrix to matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	Inverse matrix to matrix $\mathbf{A}$
$\mathbf{a}_{j,*}$	j-th row vector of matrix $\mathbf{A}$
$\mathbf{a}_{j,*}^T$	j-th row vector of matrix $\mathbf{A}$ transposed to column vector
$\mathbf{a}_{*,j}$	j-th column vector of matrix $\mathbf{A}$
$\mathbf{a}_{*,j}^T$	j-th column vector of matrix $\mathbf{A}$ transposed to volumn vector
Y^X	for sets X and Y, a set of all projections $X \to Y$
2^X	power set of X, that is $\{X' \mid X' \subseteq X\}$

Dedicated Symbols

?	Unobserved value in the rating matrix.
I	Totally ordered set of items, or something referring to the set of items when used as subscript or superscript
U	Totally ordered set of users, or something referring to the set of users when used as subscript or superscript
i, j, ℓ	Either item $i, j, \ell \in I$, or natural number $i, j, \ell \in \mathbb{N}$ used as an index in iteration or expressing arbitrary index
$\mathbb{R}^{U \times I}$ $\left(\mathbb{R}^{U \times I} \cup \{?\}\right)^{U \times I}$ $\{0,1\}^{U \times I}$ $\{0,1,?\}^{U \times I}$	Rating matrix of dimensions $\lvert U \rvert \times \lvert I \rvert$. We are omitting the cardinalities ($\lvert U \rvert \times \lvert I \rvert$) to simplify the notation, exploiting the fact that the sets are totally ordered.
x_i	i-th element in vector $\mathbf{x} \in \mathbb{R}^{\lvert I \rvert}$ for item $i \in I$, exploiting the fact the set of items I is totally ordered OR i-th element for column vector $\mathbf{x} \in \mathbb{R}^n$ if $i \in \mathbf{N}$
x_u	The u-th element for user $u \in U$ in vector $\mathbf{x} \in \mathbb{R}^{\lvert U \rvert}$, exploiting the fact the set of user U is totally ordered
$\mathbb{R}^{(U \times I)}$	Projection $U \times I \to \mathbb{R}$. The outer brackets in $(U \times I)$ explicitly denote that this is a projection, not a rating matrix (although both can be equally thought as value assignments to all $(u, i) \in U \times I$ pairs)

Miscellaneous Abbreviations

ALS Alternating Least Squares
ARs Association Rules
CB Content-Based
CF Collaborative Filtering
ItemKnn Item-Based k-Nearest Neighbors
LSH Locality-Sensitive Hashing
MF Matrix Factorization
SGD Stochastic Gradient Descent
UserKnn User-Based k-Nearest Neighbors

Chapter 1

Introduction

In this chapter, we will give brief introduction to this thesis, including motivation, problem statement, goals, and structure of the thesis.

1.1 Motivation

Recommender Systems (RSs) are important research subject in the areas of Data Mining and Machine Learning. Thanks to the huge growth in E-Commerce, Social Networks and interactive media services, it becomes very important to navigate users to content which is valuable for them. Due to huge increase in the amount of the content which is available through the use of electronic services, RSs are important tool to simplify the navigation.

The aim of RSs is to provide *users* with recommended *items* that are likely to be relevant, enjoyable, or interesting to them. Typically, such systems collect the *interactions* made by users in the past, to build recommendations at present. The interactions may include detail views, bookmarks, cart additions, purchases, plays, reads, explicit ratings, etc. Basically, the recommendation problem may be viewed as predictive modeling task.

In the recent years, large number of recommendation algorithms have been proposed and researched. These include primarily algorithms of Collaborative Filtering (CF) based on analyzing behavioral patterns across the whole userbase and making the recommendations based on clustering or searching for similarities and hidden relations between items, users, or both. The research in CF algorithms became particularly popular in 2007 thanks to a $1,000,000 Netflix Prize challenge, leading to enormous effort and attention being put by scientists into the area. But even after the competition, thanks to big commercial success of RSs, the research has been ongoing until now, resulting in more and more algorithms being proposed in the past decade.

Hand in hand with the success of RSs in everyday online business, there formed a huge and constantly growing gap between the desired practical goals of such systems, and the way how the systems are designed and evaluated in the research community.

The research community traditionally modeled the systems and their success criteria based on rating prediction, namely predicting the "number of stars" that users would assign to individual items. These approaches, however, require explicit rating datasets containing such assignments as the input from users. It appeared that it is not easy (and sometimes impossible) to collect data for such tasks in practice, because on modern Internet websites, it is only the implicit ratings what is available. The only information provided is that "user u viewed item i at time t", completely missing the numeric value expressing u's true opinion on i.

This contributes to the research-industry gap: publicly available research datasets largely differ from privately held data in the industry. Even as of today, it is not uncommon that newly published algorithms are designed for and evaluated on rating prediction tasks, including numeric accuracy measures, such as *RMSE*, as the evaluation criterion, which puts obstacles for practical deployment in many cases.

To reduce the gap between research and industry, part of the research community shifted the focus on implicit feedback datasets. In vast majority of cases, this meant adaptation of existing algorithms to such datasets. Also, novel and more practical measures, such as the *recall*@N and *precision*@N, were introduced.

The positive trend is that these novel approaches succeeded and have been adopted by biggest players in the industry. Despite that, however, is appears that the research-industry gap is still present. Specifically, it appears that measures such as the *recall*@N and *precision*@N are not enough. These measures do not reflect other desired characteristics of the recommendations.

Frequently overlooked, yet very important quality measure, is how **long-tail or niche** the recommended items are. It appeared that recommending highly popular items often maximizes measures like *recall*@N, but such recommendations are not desired nor valuable. The aim of modern RSs is to recommend long-tail content that the users would not discover otherwise. This is an important topic that needs to be addressed to further reduce the research-industry gap, with this thesis aiming to significantly contribute to it. We will revisit the existing algorithms and explore ways how to control the "long-tailness" in unified manner, together with introducing improved measures to make individual models comparable with respect to this goal.

1.2 Problem Statement

In industrial practice, it is desirable to offer users long-tail recommendations, allowing them to discover niche content precisely corresponding to their tastes. While most of the current recommendation algorithms take this into account as a general assumption, there has been very few research work seriously addressing the topic.

There is not only a proper methodology needed to evaluate how long-tail the recommended items are, but also a systematic way of controlling the long-tailness of individual models is highly desirable. This is the main problem we are going to address in this thesis.

Traditional approaches, understanding the recommendation problem as rating prediction on explicit rating datasets, with quality measured using numeric accuracy such as Mean Absolute Error (MAE) or RMSE (Root Mean Square Error), completely ignore the long-tail recommendation goal. In fact, several studies have shown that there isn't even desired correlation with measures like precision and recall.

Modern approaches, understanding recommendation as scoring the items to get Top-N candidates, with quality measurable using precision or recall, ignore the long-tail problem as well. They are unable to distinguish between models that achieve their accuracy by recommending highly popular items, and models which achieve the (possibly same level of) accuracy by recommending long-tail, niche content. To address this issue, novel evaluation methodology is needed.

Similarly, traditional models are missing systematic way of controlling how long-tail the recommended items are. Most of the models come with some "default" behavior, which is not properly investigated, simply because irrelevant measures (RMSE or pure precision or recall) are optimized. There hasn't been enough focus on how individual model hyperparameters affect the long-tailness. A systematic review of individual algorithm with the long-tail criterion in mind, is needed as well.

We will address the issues in this thesis.

1.3 Related Work/Previous Results

A very good work seriously addressing the long-tail topic is "Item Popularity and Recommendation Accuracy" [101] by Harald Steck. This thesis partially follows on this work. The author discusses the importance of long-tail recommendations, and presents novel method for training one specific type of model (Matrix Factorization) with high value of long-tail items in mind. Together with that, he proposes a modified version of *recall@N*, called the *popularity-stratified recall*. Most importantly, this approach proposes a novel parameter, named $\beta \in [0, 1]$, expressing desired bias of the model towards the long tail. This parameter is accepted by both the learning algorithm for training the model, and the popularity-stratified recall as the evaluation criterion.

Several other studies have been published as well. In [78], the long-tail recommendation problem has been identified, proposing partitioning the itemset into head and tail parts when training the model, empirically finding middle of the range as a good splitting point, but still using MAE and RMSE as evaluation criteria. In [108], the long-tail problem is identified as well, proposing the catalog coverage (called "diversity" in the paper) as additional criterion. A suite of novel graph-based algorithms for the long-tail recommendation is proposed. In [105], a distance-based and popularity-based novelty measures are proposed, but missing proper suggestions how to optimize the models for the criteria.

In this thesis, we build on these previously published results with the aim measuring and systematically controlling the magnitude of long tail recommendation.

1.4 Goals of the Dissertation Thesis

The main goals of this thesis are as follows:

1. Review the existing recommendation learning algorithms and models proposed by various researchers over time.
2. Unify the algorithms under common framework.
3. Introduce systematic way how to control the magnitude of long-tail recommendations for all the presented algorithms.
4. Introduce a novel evaluation methodology that would measure both the recommendation accuracy and the magnitude of long-tail recommendations.
5. Prove that the introduced theoretical concepts really work and expected, including not only publicly available datasets, but also datasets taken from industry.

1.5 Structure of the Dissertation Thesis

The thesis is organized into 5 chapters as follows:

1. *Introduction*: Describes the motivation behind our efforts together with our goals.
2. *Background and State-of-the-Art*: Introduces the reader to the necessary theoretical background and surveys the current state-of-the-art. This includes overview of data sources in RSs, recommendation tasks, existing algorithms and models, as well as methods of evaluation.
3. *Overview of Our Approach*: Presents our approach to controlling and measuring the magnitude of long-tail recommendations. We propose generalized popularity regularization, manipulating model capacity for most of the algorithms researched in the past two decades, and introduce a framework for multi-objective model optimization, searching for Pareto-optimal front in the recall-coverage plane.
4. *Main Results*: Demonstrates that theoretically predicted behavior of the proposed capacity-controlling hyperparameters for individual algorithms really corresponds to empiric behavior on several datasets.
5. *Conclusions*: Summarizes the results of our research, suggests possible topics for future research, and concludes the thesis. There is also a list of contributions of this dissertation thesis.

Chapter 2

Background and State-of-the-Art

In this chapter, we will give systematic introduction to the area of Recommender Systems (RSs). We will describe the most important type of data sources used by these systems, and explain how the data from such sources can preprocessed into standard forms (most importantly, the rating matrices). Further on, we will define the most common tasks solved by such systems, followed by overview of the most prominent approaches to solve them. We will explain how hyperparameterized learning algorithms are used to produce recommendation models, and how the quality of these models can be evaluated. The algorithms are introduced under unified framework, which allows the resulting models being used for different recommendation tasks, as well as being systematically evaluated and controlled by the hyperparameterization in the following chapters.

2.1 Data Sources in Recommender Systems

Recommender Systems can be classified as wide subfield of Machine Learning (ML), making them heavily reliant on the provided data. They typically work with the following types of data sources:

- interaction data,
- item attributes data,
- user attributes data.

The cornerstone of any recommendation dataset is an ordered set of **items**, $I = \{i_1, \ldots, i_n\}$. Items refer to recommendable entities, such as products in E-Commerce, multimedial content, advertisements, news articles, etc. The items may have **item attributes** of different types. These attributes are often referred to as the Content. It may include names, categorizations, textual description, or even images, audio data, video data, etc.

The purpose of Recommender Systems is to serve to the users. In this thesis, the (ordered) set of users will be denoted as $U = \{u_1, \ldots, u_m\}$. Same as items, also users may have various **user attributes**. In practice, these attributes may include demographical data, such as gender, age, or geographic location.

An important component of most recommendation datasets are also the user-item interactions. They typically describe events performed by the users on the items, such a viewing an item's detail page, purchasing an item, or putting it to favorites.

2.1.1 Attribute Data

Let us denote the set of item attributes $A^I = \{A_1^I, \ldots, A_r^I\}$. For some datasets, A^I may be empty if the attributes are missing. We will denote $a_j^I \colon I \to \mathrm{dom}(A_j^I)$ the function that assigns the value of attribute A_j^I to items from I, and $a^I \colon i \mapsto \left(a_1^I(i), \ldots, a_r^I(i)\right)$ the function that assigns the values of all the attributes.

The domains $\mathrm{dom}(A_j^I)$ of individual attributes may be of various types and flavours:

2.1.1.1 Unstructured text

Given as arbitrary-length sequence from Σ^*, where Σ is an alphabet of some natural language, such as English. Besides letters, Σ may include also digits, punctation marks, and other characters, which typically induces a need of preprocessing. Sparse vectors of words from a common dictionary may be produced by techniques like tokenization and stemming (bag-of-words approach), but also dense vectors are possible, for example by application of a dimensionality-reducing algorithm either on top of bag-of-words (e.g. SVD), or by an embedding technique such as word2vec.

2.1.1.2 Tags, Categorization

Subsets of some finite set of tags/categories that each number is assigned/member of. Assuming the tags/categories are expressed in some alphabet Σ, the $\mathrm{dom}(A_j^I)$ can be thought as 2^{Σ^+}.

2.1.1.3 Numbers

Typically natural numbers from $\mathbb{N}$ or real numbers from $\mathbb{R}$ describing items in various numerical measures (number of CPUs for a server, duration in minutes for a movie/song).

2.1.1.4 Image Data

In the simplest case, images are given as 2D rasters. Frequently used encoding works with elements from $\{0, \ldots, 255\}^{h \times w}$ for grayscale and $\left(\{0, \ldots, 255\}^{h \times w}\right)^3$ for RGB pictures. Such a raw image typically requires preprocessing, such as color histogram extraction or embedding techniques using convolutional neural networks.

Besides attributes of items, there may also be set $A^U = \{A_1^U, \ldots, A_s^U\}$ of user attributes provided. Symmetrically to items, there is a function $a_j^U : U \to \mathrm{dom}(A_j^U)$ which assigns value of one attribute, and $a^U : u \mapsto \left(a_1^U(u), \ldots, a_s^U(u)\right)$ which assigns all the attributes.

2.1.2 Tokenized Attribute Data

Considering attribute data and their various types introduced in 2.1.1 with many unstructured options such as raw texts, it is sometimes useful to preprocess them into a unified form. One of the possibilities is creating a set of *tokens* by means of process called the *tokenization*. A *token* can be though as a unique string in Σ^+ for some alphabet Σ, and tokenization can be thought as a function $\textit{tokenization}: \mathrm{dom}\left(A_1^I\right) \times \ldots \times \mathrm{dom}\left(A_r^I\right) \to 2^{\Sigma^+}$ $\left(\text{or } \textit{tokenization}: \mathrm{dom}\left(A_1^U\right) \times \ldots \times \mathrm{dom}\left(A_s^U\right) \to 2^{\Sigma^+} \text{ in case of users}\right)$ that is able, for each item $i \in I$ (or each user $u \in U$), based on its attribute values $a^I(i)$ (or $a^U(u)$), to produce a set of tokens extracted from these values. There are many possible implementations of *tokenization*, but here is the one to be used in this thesis. It can be described as follows:

- Start with empty set: `tokens` $\leftarrow \emptyset$.
- For each **string** attribute value, apply tokenization (to split sentences into words using delimiters such as space, dot, comma, etc.) and stemming for given language, and extend `tokens` with tokens in form `<attribute-name>=<value>` where `<value>` is a stemmed version of each word, excluding stopwords.
- For each attribute value of type **set**, extend `tokens` with tokens in form `<attribute-name>=<element>` for each `<element>` in the set.
- For each integer attribute value, extend `tokens` with token in form `<attribute-name>=<value>` where `<value>` is a string representation of the integer value.
- For each boolean attribute value, extend `tokens` with token `<attribute-name>=<value>`, where `<value>` is either `true` or `false`.

For example, as the result of *tokenization* on a sample "Back to the Future Part III" item from the MovieLens dataset [43], we may obtain tokenset of

$$\{\texttt{title=back}, \texttt{title=futur}, \texttt{title=part}, \texttt{year=1990}, \\ \texttt{genres=Sci-Fi}, \texttt{grenres=Comedy}, \texttt{grenres=Western}\}$$

2.1.3 Interaction Data

Besides attribute data about items and/or users, in many practical situations, the most important source of data are **user-item** interactions. They encode some actions performed by users on items.

The most commonly available type of interaction data are records that can be easily extracted from existing server logs or purchase database:

Detail-Views Occurring when user views a product detail page in E-Commerce system, position description in job advertisements HR portal, or movie description in a Video-on-Demand system.

Cart-Additions Occurring in E-Commerce systems made by users that were attracted by the items enough to put them to a shopping basket.

Bookmarks In a video-streaming service, discussion forums, or classifieds portals, they mean that a given user is interested in a given item and wants to come to it later or when they have more time or the item becomes available.

Plays Natural events in an audio or video streaming services.

Likes, Shares Concepts known from contemporary social networks. If there is only positive rating possible (with no dislike option), we also consider these events as implicit interactions.

Purchases Common events in E-Commerce system or classifieds portal.

It should be obvious there is a lot of other possible types of such actions, and that they are easy to collect, simply because there is no extra effort needed from the side of users, and often even not from the side of the service provider. Such interaction data are a side product of normally using the service, and they might not have been originally intended as a data source for a Recommender System. For this reason, we will refer to this types of interactions the **implicit interactions**.

Implicit interactions may be thought as a set $Y = \{y_1, \ldots, y_k\}$, where $y_j = (u_j, i_j, t_j, d_j)$ meaning:

- $u_j \in U$ is the user who did the interaction,
- $i_j \in I$ is the interacted item,
- $t_j \in \mathbb{R}$ encodes the timestamp when the interaction happened,
- d_j may be arbitrarily complex additional information on the interaction, such as:
  - type of the interaction,
  - duration of the detail-view in seconds,
  - portion of the audio/video played expressed as relative ratio or in seconds,
  - amount of the item purchased or put to a shopping cart,
  - price and currency in which the given amount of the item has been purchased or put to a shopping cart.

Besides implicit interactions possibly collected just by running the service, there are also **explicit interactions** possible, allowing users to explicitly express their preference on the given item. These may include:

2.1.3.1 Ratings in Numbers of Stars

This is a basic concept from traditional literature on Recommender Systems. An item may be assigned e.g. 1-5 stars, where 1 star means strictly negative opinion and 5 stars mean best possible opinion.

2.1.3.2 Likes and Dislikes.

In situations where both of these actions are possible, we are also talking about explicit interaction. Similarly to rating in number of stars, both positive and negative opinion can be expressed.

Assuming implicit interactions give us information about likely positive bias of a user towards the interacted item, but there is a lot of uncertainty. Specifically, there may be a lot of false positives emerging from users who started playing a video/song, but found they don't like it, or made a purchase in an E-Commerce platform, but weren't satisfied when the goods arrived by post. Explicit interactions, on the other hand, come with much higher certainty: we may assume that they represent user's true opinion at the time of rating being performed.

Explicit ratings may be again thought as a sequence of events occurring over time. Let us denote them $X = \{x_1, \ldots, x_\ell\}$, where $x_j = (u_j, i_j, t_j, g_j)$ meaning:

- $u_j \in U$ is the rating user,
- $i_j \in I$ is the rated item,
- $t_j \in \mathbb{R}$ is the time when the interaction happened,
- $g_j \in [-1, 1]$ is the rating (grade) that the user assigned to the item.

Using $g_j \in [-1, 1]$ is just for convenience since any rating scale can be easily converted to this interval. In case of likes and dislikes, likes can be encoded as $+1$ and dislikes as -1. In case of N-star rating scale from 1 to $N \geq 2$ stars and given rating g'_j on that scale, g_j can be obtained as $\frac{N-g'_j+1}{1-N}$. Despite raw explicit ratings g'_j are processed directly in some literature, most of the contemporary research assumes different normalization methods, and mapping to $[-1, 1]$ does not prejudice the generality.

2.1.4 Rating Matrices

Native streams of implicit and explicit interactions provide a lot of information. Most notably, thanks to their temporal component, they can capture how preferences of users

evolve over time, both for individuals and for the userbase as a whole. One particular user may search for one type of product, and while she is searching, similar products to those viewed should be recommended to her. But once purchased, the product may serve her for years, and she may became interested in completely different types of products. Similarly, the user may enjoy certain music genres, but may switch to slightly different genres over time. Furthermore, such phenomena may not only affect individual users. In domains like fashion, it is natural that different types of clothes are trending during summer and winter. In popular music, new music hits are continuously published, and yet unknown artists are becoming famous.

Last, but not least, different interaction types bring information of completely different nature. Models of Collaborative Filtering, described later in this thesis in Sec 2.3.2, often work with concepts such as "frequently interacted together". But consider user searching for a refrigerator. Such a user may view details of multiple refrigerators, studying their different characteristics while hesitating. But once the final decision is made, the users puts into the shopping cart and orders at most one refrigerator. So while multiple refrigerators are frequently viewed together, they are never purchased together.

Despite raw sequences of interactions provide very rich source of information, many popular methods in Recommender Systems assume transformation to more practical form. The clearly most popular is the so-called **rating matrix**. It is simply a matrix $\mathbf{R} \in \mathbb{R}^{|U|\times|I|}$ with each element $r_{u,i}$ representing a condensed information about the preference of user $u \in U$ on item $i \in I$. Temporal information, same as information about interaction types, is ignored. From the aforementioned reasons, this brings several drawbacks, but having data as a real-valued $|U|\times|I|$ matrix also brings many advantages. Most notably, we get notions, concepts, and algorithms from linear algebra, which have been developed for decades. For this reason, we will adopt the rating matrix as the main data source in this thesis. Although other methods, such as next-in-sequence-prediction using e.g. Recurrent Neural Networks [45, 99] or Sequential Patterns Mining [4, 75] (which is even studied and measured in a Bachelor thesis supervised by the author of this thesis [7]), we will consider out of the scope of this thesis.

For now, let us briefly discuss the possibilities of building the rating matrix from the raw interaction data.

The general algorithm is to instantiate an empty matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{|U|\times|I|}$, containing only ?s at the beginning, where ? is a special value denoting the value in unknown or is unobserved. Based on the set of interactions (implicit, explicit, or both), ? s are replaced at positions where some interaction took place, using different aggregation methods to obtain specific values from $\mathbb{R}$.

Because of the sizes $|U|$ and $|I|$ may easily go to millions, and because each user typically interacts with only very limited subset of items, the rating matrix is typically **extremely sparse**, meaning $|\{r_{u,i} = ?|u \in U, i \in I\}| \gg |\{r_{u,i} \neq ?|u \in U, i \in I\}|$.

2.1.4.1 Explicit Ratings

The simplest approach to fill the data matrix is to simply take solely the explicit ratings. This approach became popular thanks to the popular Netflix Prize organized by Netflix, Inc., in 2006-2009, and also thanks to MovieLens dataset, maintained since 1997 by GroupLens Research, a research laboratory in the Department of Computer Science and Engineering at the University of Minnesota. Both these datasets are frequently studied by the Recommender Systems research community, and offer explicit interactions compliant with the definition of $X = \{(u_1, i_1, t_1, r_1), \ldots, (u_\ell, i_\ell, t_\ell, r_\ell)\}$ in 2.1.3.

To fill the rating matrix, for each user-item pair, we may simply take the last explicit rating provided, if any exists, or with ? otherwise:

$$r^E_{u,i} = \begin{cases} g_j\colon & (u_j, i_j, t_j, g_j) \in X \wedge \\ & \forall (u_k, i_k, t_k, g_k) \in X\colon t_k \leq t_j & \text{if } \exists (u_k, i_k, t_k, g_k) \in X\colon u_k = u \wedge i_k = i \\ ? & & \text{otherwise} \end{cases} \tag{2.1}$$

Average instead of the last value for each user-item pair can be taken as well, but there is not much difference, because for vast majority of user-item pairs there is at most one explicit rating in common datasets.

It is noteworthy that matrices generated from explicit ratings typically consist of whole range of different values, including numbers that are negative, encoding negative preferences. This allows us to build algorithms and evaluation metrics that take negative ratings into account. The disadvantage is, however, general absence of explicit ratings in real-world datasets, which creates a barrier between some results that come from academia, and the algorithms actually applicable in industry.

2.1.4.2 Implicit Ratings of Certain Types

While explicit ratings are often difficult to collect to the extent that would cover reasonable portion of items or users, implicit ratings are often much more accessible and cheaper data source. As mentioned in 2.1.3, different interactions types may sometimes have different characteristics and might not correlate. Items "frequently viewed together" may differ from items "frequently bought together". While viewing multiple bikes may be common, purchasing multiple bikes would be rather exceptional. User will buy a helmet or a pump together with a bike much more likely that another bike.

This leads to construction of implicit interaction matrices for different purposes.

Considering the set of implicit interactions $Y = \{(u_1, i_1, t_1, d_1), \ldots, (u_k, i_k, t_k, d_k)\}$ as introduced above, we can easily define filtering function $\varphi\colon Y \to \{0, 1\}$ and weighting function $w\colon Y \to \mathbb{R}$ to construct the rating matrix from implicit interactions

$$r^I_{u,i} = \Gamma\left(\{(u_j, i_j, t_j, d_j) \in Y | \varphi\left((u_j, i_j, t_j, d_j)\right) \wedge u_j = u \wedge i_j = i\}\right) \tag{2.2}$$

where $\Gamma: 2^Y \to \mathbb{R} \cup \{?\}$ combines multiple interactions for the same user-item pair into one number. For example, if $\forall y \in Y: 0 \leq w(y) \leq 1$, one reasonable implementation of Γ can be:

$$\Gamma(Y') = \begin{cases} \min\left(\sum_{y' \in Y'} w(y'), 2\right) & \text{if } Y \neq \emptyset \\ ? & \text{otherwise} \end{cases} \tag{2.3}$$

As a real-world example, consider we want to construct rating matrix based only on additions to a shopping cart and purchases. Then we use filtering function φ which returns 1 only for interactions of type "cart addition" and "purchase", whilst for interactions of type "detail views" or "bookmarks" it returns 0. We can also use function w which returns 0.75 for interaction of type "purchase" and 0.25 for interaction of type "cart addition", possibly also incorporating additional information attached to the interaction, such as the amount added to cart/purchased. Using the above Γ function, it is assured that multiple purchases of the same item by the same user add up to higher number, but to prevent extreme values possibly introduced by outlier users (such as legal entities performing repetitive purchases on regular basis), the upper bound for the sum is 2. This produces quite useful matrix directly applicable for construction of cross-sell models. Using similar approach and using different φ and w functions, we may focus on different types of interactions (possibly only detail-views) and weight interactions using different logic (such as taking the age of the interaction into account).

2.1.4.3 Mixed Implicit Ratings

In some domains, all the products are complementary and purchase of one of them does not contradict purchase of another. Good example of such a domain is multimedia with products such as movies, books, songs, etc. In such a domains, one can omit the filtering function φ and mix all the implicit interactions together, assigning them only different weights through w, such as $w(y) = 0.75$ if y is purchase, and $w(y) = 0.1$ if y is detail-view. Then simply:

$$r^I_{u,i} = \Gamma\left(\{(u_j, i_j, t_j, d_j) \in Y | u_j = u \wedge i_j = i\}\right) \tag{2.4}$$

It is no accident that such a type of complementary content (such as movies) has received a lot of attention in research since no constraints are imposed on the recommendation model.

2.1.4.4 Mixed Implicit and Explicit Ratings

In many real-world situations, there is both explicit and implicit feedback available. Users may browse and purchase the content, producing implicit interactions, but some users assign explicit ratings to some items from time to time. It is desirable to mix all the available data to a single rating matrix.

One possible approach, that we will follow in this thesis, is to project both implicit and explicit interactions to the same scale. By mixing the formulas 2.1 and 2.4, we can either take the explicit rating if it is known, or try to estimate it from implicit interactions if it is unknown:

$$r_{i,j}^{M} = \begin{cases} r_{i,j}^{E} & \text{if } \exists (u_j, i_j, t_j, r_j) \in X \colon u_j = u \wedge i_j = i \\ r_{i,j}^{I} & \text{otherwise} \end{cases} \tag{2.5}$$

Such a matrix still contains "?" for user-item pairs with no interactions, but otherwise contains whole scale of other values. The most frequent will typically be values coming from implicit interactions, but explicitly positive or negative ratings are also present.

2.2 Recommendation Tasks

The aim of Recommender Systems is to process the existing input data (rating matrix, user attributes, item attributes) in a way that predictions can be made for unknown cases. This fits the general definition of supervised learning. In context of recommendation, the goal of such learning is simple: knowing true preferences of some users to some items, what is the predicted preference for an unknown user-item pair? Knowing that, recommendations of possibly interesting, previously unseen items, can be generated to users.

Despite the nature of recommendation tasks varies, we may consider there is always a two-phase process:

1. Existing data, sometimes called as the training data, are passed as input to the **learning algorithm**, which produces a model as the output.

2. The **model** is used to make guesses (predictions) for situations previously unseen in the data with the aim of generating useful recommendations.

2.2.1 Learning Algorithms and Hyperparameterizations

The **learning algorithm** $\mathcal{A}$, sometimes also referred to as the *training algorithm* (we will consider the terms *training* and *learning* as synonyms throughput this thesis), can be thought as higher order function: It takes data and produces a model, which is itself a function. In this thesis, all the discussed algorithms are strictly assumed as solving supervised learning tasks. That is, in general, given set of k training examples $\mathcal{T} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_k, \mathbf{y}_k)\}$, where $\forall \ell \in \{1, \ldots, k\} \colon \mathbf{x}_\ell \in \mathcal{X} \wedge \mathbf{y}_\ell \in \mathcal{Y}$ for some abstract sets $\mathcal{X}$ and $\mathcal{Y}$, the algorithm $\mathcal{A}$ produces a **model** $m \colon \mathcal{X} \to \mathcal{Y}$. We denote this as

$$\mathcal{A} \colon 2^{\mathcal{X} \times \mathcal{Y}} \to \mathcal{Y}^{\mathcal{X}} \tag{2.6}$$

where $2^{\mathcal{X} \times \mathcal{Y}}$ is the set of all possible training datasets ($\mathcal{T} \subseteq \mathcal{X} \times \mathcal{Y}$ as introduced above), and $\mathcal{Y}^{\mathcal{X}}$ denotes set of all projections from $\mathcal{X}$ to $\mathcal{Y}$. Besides the training data, the learning algorithm often takes some parameterization $P \in \mathcal{P}_{\mathcal{A}}$ that influences the learning process

and hence the resulting model. In the literature, $\mathcal{P}_\mathcal{A}$ is often referred to as *space of hyperparameters* [22], and specific $P \in \mathcal{P}$ as the *hyperparameterization* [31, 103]. This leads to improved general definition of learning algorithm as

$$\mathcal{A}: \mathcal{P}_\mathcal{A} \times 2^{\mathcal{X} \times \mathcal{Y}} \to \mathcal{Y}^\mathcal{X} \tag{2.7}$$

Hyperparameterization may describe various characteristics that will be discussed for individual models later on in this thesis, but they typically include learning rates, regularizations, dimensions of latent feature spaces, neighborhood sized, number of layers in neural networks, etc.

In the rest of Sec. 2.2, we will introduce specific recommendation tasks and introduce specific examples of $\mathcal{X}$ and $\mathcal{Y}$ for these tasks. Later on in Sec. 2.4, we will introduce specific learning algorithms, that we will refer to are **recommendation algorithms** throughout this thesis.

2.2.2 Rating Prediction

For rating matrices built from explicit ratings (and henceforth encoding both positive and negative ratings), one of frequently studied tasks in the **rating prediction** [63, 102, 85]. Assuming that the opinion of each user is expressible and a real number from some scale (and also that this opinion is basically fixed and does not evolve), there is a natural supervised learning task: given rating $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{U \times I}$ and an unobserved user-item pair (u, i) (such that $r_{u,i} = ?$), predict $\hat{r}_{u,i} \in \mathbb{R}$ as an estimate of the explicit rating that u would assign to i is asked to. Learning Rating Prediction model can be thought as searching for $m^{\text{RP}}: U \times I \to \mathbb{R}$ which predicts $\hat{r}_{u,i} = m(u, i)$ given set of training examples $\mathcal{T} \subseteq U \times I \times \mathbb{R}$.

Rating Prediction has been for years one of the most researched topic in the area of Recommender Systems for multiple reasons. One reason to mentioned is the aforementioned Netflix Prize run in years 2006-2009, which caught attention of lot of researchers thanks to the grand prize of $1M [10, 12]. But also other datasets similar to Netflix Prize have been released earlier in scientific circles, most notably the MovieLens datasets [43]. Furthermore, since explicit rating are from continuous scale, the task is mathematically elegant and various numeric error-minimizing optimization approaches may be used.

The task of Rating Prediction can be states as follows: Given rating matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{U \times I}$ (equivalent to $\mathcal{T} = \{(u, i, r_{u,i}) \mid u \in U, i \in I, r_{u,i} \neq ?\}$), construct a model $m^{\text{RP}}: U \times I \to \mathbb{R}$ which predicts the rating value for every (u, i) pair. We are hence searching for learning algorithm $\mathcal{A}^{\text{RP}}$ and its hyperparameterization $P \in \mathcal{P}_{\mathcal{A}^{\text{RP}}}$ that will construct such a model:

$$\mathcal{A}^{\text{RP}}: \mathcal{P}_{\mathcal{A}^{\text{RP}}} \times (\mathbb{R} \cup \{?\})^{U \times I} \to \mathbb{R}^{(U \times I)} \tag{2.8}$$

The behind idea is that Rating Prediction can be actually used as an underlying tool for tasks such as Top-N recommendation (described in Sec. 2.2.4 below). Having model m^{RP} and given user u, one may simply predict $\hat{r}_{u,i}$ for all the items $i \in I$ and use the N highest scored by m^{RP}.

2.2.3 Binary Classification

Another popular task in Recommender Systems, slightly closer to practice than rating prediction, is the **binary classification** [8, 44, 100]. It builds on top of assumption that for a given user u, each item can be either classified as relevant (liked, candidate for purchase) or irrelevant (disliked, uninteresting).

For Binary Classification tasks, the rating matrix $\mathbf{R}^B$ is from $\{0, 1, ?\}^{U \times I}$. Given rating matrix created using one of the approached described above (that is taking explicit $\mathbf{R}^{\mathrm{E}}$, implicit $\mathbf{R}^{\mathrm{I}}$, or mixed $\mathbf{R}^{\mathrm{M}}$, all referenced as $r_{i,j}$ in the below formula), one can easily construct $\mathbf{R}^B$ using a well-chosen threshold $\theta \in \mathbb{R}$:

$$r^{\mathrm{B}}_{i,j} = \begin{cases} 1 & \text{if } r_{i,j} \neq ? \wedge r_{i,j} \geq \theta \\ 0 & \text{if } r_{i,j} \neq ? \wedge r_{i,j} < \theta \\ ? & \text{if } r_{i,j} \neq ? \end{cases} \tag{2.9}$$

Binary Classification task in Recommender Systems can then be stated as follows: Given matrix $\mathbf{R} \in \{0, 1, ?\}$ (equivalent to $\mathcal{T} \subseteq \{(u, i, r_{u,i}) \mid u \in U, i \in I, r_{u,i} \neq ?\}$), construct a model $m^{\mathrm{BC}}\colon U \times I \to \{0, 1\}$ which predicts whether given item $i \in I$ is relevant for user $u \in U$ for each (u, i) pair. To accomplish that, we are searching for binary classification learning algorithm $\mathcal{A}^{\mathrm{BC}}$ that, provided hyperparameterization $P \in \mathcal{P}_{\mathcal{A}^{\mathrm{BC}}}$, constructs such a model:

$$\mathcal{A}^{\mathrm{BC}}\colon \mathcal{P}_{\mathcal{A}^{\mathrm{BC}}} \times \{0, 1, ?\}^{U \times I} \to \{0, 1\}^{(U \times I)} \tag{2.10}$$

For given user, we want to find exactly those items that are of her interest. In practice, this task makes sense only for relatively small sets of relevant items, because it impossible for a human user to browse all the relevant items it their number goes to millions. For such scenarios, Top-N recommendation is much more suitable.

2.2.4 Top-N Recommendation

Top-N Recommendation task [51, 26, 23] if by far most important and widespread recommendation task in industrial practice [66, 25, 27], and is henceforth the main subject of focus in this this work. Given $N \in \mathbb{N}, N \ll |I|$ attempts (or simply slots for items to be displayed), and user $u \in U$, our goal is to find N items which are the most relevant for current user. Real-world use-cases of Top-N Recommendation include different scenarios such as displaying recommended items at different places in the user interface, putting recommendation to personalized emails, etc.

The rating matrix may remain unchanged for the task. The only assumption is that for the numbers it contains (no matter how they were constructed from the interactions), it hold that higher is better, that is more relevant.

Top-N Recommendation task may be stated as follows: Given rating matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{U \times I}$ (inducing $\mathcal{T} \subseteq \{(u, i, r_{u,i}) \mid u \in U, i \in I, r_{u,i} \neq ?\}$), construct a model $m^{\text{Top-}N}$:

$U \to \{I' \subset I \mid |I'| = N\}$. We are searching for a hyperparameterized algorithm $\mathcal{A}^{\text{Top-}N}$ able to construct such a model:

$$\mathcal{A}^{\text{Top-}N}: \mathcal{P}_{\mathcal{A}^{\text{Top-}N}} \times (\mathbb{R} \cup \{?\})^{U \times I} \to \{I' \subset I \mid |I'| = N\}^U \tag{2.11}$$

There may be logical requirement added not to recommend the items for which the rating is already known, so that it holds

$$\forall (u, i) \in U \times I: r_{u,i} \neq ? \implies i \notin m^{\text{Top-}N}(u) \tag{2.12}$$

We will strictly follow this assumptions also in thesis. It comes from practical requirement to only recommend items that are likely to be new for the users, helping them discover content they wouldn't discover by themselves otherwise.

2.2.5 Learning to Rank

The idea of Learning to Rank [83, 97, 50] comes from Information Retrieval [67]: searching for a permutation of all the items from I sorting the items according to their relevance in the given context. While in Information Retrieval, the context is typically given by a search query [24], in Recommender Systems, the context is given by user $u \in U$.

Same as in case of Top-N recommendation, rating matrix of any kind can be used for this task as long as higher $r_{u,i}$ means higher relevance of item i for user u.

Learning to Rank in Recommender System can be stated as follows: Given rating matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{U \times I}$ construct a model $m^{\text{Rank}}: U \to \mathfrak{S}(I)$, where $\mathfrak{S}(I)$ denotes the set of all permutations of items from I. To be able to build such a model, we are searching for a learning algorithm $\mathbf{A}^{\text{Rank}}$ hyperparameterized by $\mathcal{P}_{\mathcal{A}^{\text{Rank}}}$:

$$\mathcal{A}^{\text{Rank}}: \mathcal{P}_{\mathcal{A}^{\text{Rank}}} \times (\mathbb{R} \cup \{?\})^{U \times I} \to \mathfrak{S}(I)^U \tag{2.13}$$

One practical disadvantage of ranking is that sorting all the items if usually not necessary and may hence require much higher computational resources. Considering millions of items, the changing the order of the trailing 999,000 items often does not have significant impact, since it's out of the user's capacities to go through the whole sorted list.

2.2.6 Other Recommendation Tasks

There are multiple other recommendation tasks that go beyond the extent of this thesis. Let us just briefly introduce them to make a more complete picture of how broad the area of Recommender Systems is.

2.2.6.1 Related Items Recommendations

One of the simplest yet very frequent use-cases of Recommender Systems is to recommend items somehow related to one particular item. When user is viewing an item on a website, it

is often desirable to offer her alternative items that she might also like, possibly increasing average session duration, increasing customer lifetime value, loyalty, etc.

While all the above tasks of Top-N Recommendation and Learning to Rank were defined as user-oriented, that is building the recommendations in context of given particular user $u \in U$, they can be symmetrically defined in context of a particular item $i \in I$ instead. This leads to searching for models $m_{\mathrm{II}}^{\text{Top-}N}: I \to \{I' \subset I \mid |I'| = N\}$ for item-based Top-N Recommendation, and $m_{\mathrm{II}}^{\mathrm{Rank}}: I \to \mathfrak{S}(I)$ for item-based Ranking.

From the two tasks mentioned, the Top-N Recommendation is the one giving most practical sense, and there is typically limited space for related items to be presented next to the one currently viewed.

2.2.6.2 Recommending Related Users

It is not only items that can be recommended for one particular user $u \in U$. Related users can be recommended as well. Same as items may be related to items in different ways (purchases together $\rightsquigarrow$ compatible, viewed together $\rightsquigarrow$ similar), the same hold for users.

For example, consider an HR system where the users are candidate employees viewing different job advertisements. An HR specialist may browse users, and for one particular users = job candidate, her goal is to find similar users = candidates. Users interested in (interacting with) similar content are likely to be similarly skilled and hence being also good candidates for the position that the HR specialist is looking to occupy by and employee. Another example can be online gaming portal, where players are searching for game opponents or teammates interested in similar games. Both these examples lead to searching users that are is some way very similar to one particular user $u \in U$.

As the opposite example, consider a social network with online dating functionality. In majority of cases, males and females are interested in different content, but when it comes to searching for a dating partner, males are often interested in females and vice versa. For a given user, users who are compatible (males to females) are much more desirable candidates for recommendation rather than users who are similar (males to males). This is somehow similar to recommending compatible (helmet to bike) rather than similar (bike to bike) products in a shopping cart.

Different recommendation approaches are required for both cases, and the area hasn't been researched well yet. But in general, we need models like:

- Users-to-User Top-N recommendation: $\left(m_{\mathrm{UU}}^{\text{Top-}N}: U \to \{U' \subset U \mid |U'| = N\}\right)$ or
- Users-to-User Ranking $\left(m_{\mathrm{UU}}^{\mathrm{Rank}}: U \to \mathfrak{S}(U)\right)$.

2.2.6.3 Recommending Users for Items

Users can play a role of recommendable entities not only for other users, but for items as well. Consider an online retailer who has many pieces of one particular product $i \in I$ on the stock, and wants to sell that product out due to the shortly ending season. One

option how to sell that product out is to find a relatively small subset of users who might be interested in that product, and approach them via an emailing campaign, informing about discount on that product. The motivation is not to disturb and annoy otherwise loyal users by a campaign which does not concern them.

Another example is sending personalized push notifications, such on mobile devices, about new content (news, job offers) only to carefully selected users who might be interested.

In some cases, Binary Classification as described above can be directly used for this task. In other cases, models like:

- Users-to-Item Top-N recommendation: $\left(m_{\mathrm{IU}}^{\text{Top-}N}: I \to \{U' \subset U \mid |U'| = N\}\right)$ or
- Users-to-Item Ranking $\left(m_{\mathrm{IU}}^{\mathrm{Rank}}: I \to \mathfrak{S}(U)\right)$

are more appropriate.

2.3 Recommendation Approaches

Across the literature, there are multiple criteria for classification of recommendation approaches and algorithms. The most common division is based on

- type way in which data is processed:
  - Memory-Based Recommendation,
  - Model-Based Recommendation,
- the type of data used for modeling:
  - Collaborative Filtering,
  - Content-Based Recommendation,
  - Hybrid Models.

These two criteria are practically independent as different types of data may be processed in different ways.

2.3.1 Model-Based vs. Memory-Based Recommendation

Although terminologically not exactly precise, as **Model-Based** approaches are recognized algorithms which take the data (such as tha rating matrix of item/user attributes) and process them in some complex way (referred to as the training) such that a completely new structure emerges. This new structure (referred to as the model) typically contains compressed information extracted or abstracted from the data. Some information is lost during the process as one of the goals is to compress the large amounts of data into more

compact form to allow fast recommendation after relatively long time spent on the training. Examples of such algorithms can be matrix factorization, training neural networks, or precomputing similar items in the Item-Based k-NN (all described below).

In contrast, **Memory-Based** approaches is a common name for the family of algorithms that work directly on the original data when generating recommendations, with only minimal preprocessing phase. In the area of machine learning, Memory-Based approaches are more commonly known as "Lazy learning".

Taken abstractly, both Memory-Based and Model-Based approaches comply to 2.2, where we mentioned two-phase process of building a model in the first phase and using the model for generating recommendations in the second phase. In case of Memory-Based methods, as the term "Lazy learning" suggests, the learning phase can be still considered as present, but with very trivial implementation of storing all the data for future purposes. In the second phase of generating recommendations, the model just may (or may not!) take longer to respond that in the non-Memory-Based approach.

Because different parameterizable algorithms producing both Memory-Based and non-Memory-Based models are studied closely later in this thesis, we will **not** follow this traditional Model-Based and Memory-Based division and will always consider there is model which has been produced in abstract way by an abstract learning algorithm.

2.3.2 Collaborative Filtering

Collaborative Filtering (CF) is a name for the whole family of methods which work with the rating matrix to produce the model [37, 84]. The models may be based on multiple different operations performed on top of the matrix:

- working with relationships between rows
  - measuring vector similarity between rows (User-Based k-NN)
- working with relationships between columns
  - measuring vector similarity between columns (Item-Based k-NN),
  - mining co-occurrences of groups of items within rows (Association Rules)
- compressing the matrix
  - searching for SVD-inspired decompositions (Matrix Factorization),
  - using neural auto-encoders to compress the rows or the columns (AutoRec).

All the algorithms mentioned will be introduced later in this chapter.

2.3.3 Content-Based Recommendation

In contrast to CF methods, Content-Based (CB) Recommendation rely on data types other than the rating matrix, most commonly on the item attribute data, but may work with user attribute data as well. The main ideas behind building a Content-Based recommendation algorithms lies in processing of item and user attributes in various ways to.

- Compute similarity between two items using item values as given by a^I. This typically includes attribute vectorization into $\mathbb{R}^n$ or $\{0,1\}^n$ so that similarity functions $sim\colon I^2 \to \mathbb{R}$ emerge using concepts like cosine or TF-IDF can be used. This allows us to find similar items given on particular item $i \in I$ based just on their attributes.

- Estimate usefulness of item i to user u based on item attribute values $a^I(i)$ and the attribute values $\left\{a^I(j) \middle| r_{u,j} \neq ?\right\}$ of items that the user interacted with. This leads to searching for similarity measure $\mathrm{Sim}\colon I \times 2^I \to \mathbb{R}$ between one particular item $i \in I$ and a set of interacted items 2^I, possibly taking different rating values from matrix $\mathbf{R}$ into account. Using Sim, we can find items similar to set of items interacted by u taken as a whole, and recommend them to user u.

- Estimate usefulness of item i to user u based on user attribute values $a^U(u)$ and item attribute values $a^I(i)$, not taking interactions into account at all. The similarity function is $sim\colon U \times I \to \mathbb{R}$. One practical example is recommending job advertisements to users based on job descriptions given as item attributes and resumes given as user attributes. Pairs $(u,i) \in U \times I$ where $a^U(u)$ and $a^I(i)$ contain a lot of shared keywords/skills, will have high value of $sim(u,i)$, and no interaction data is needed for this task.

2.3.4 Demographic Recommendation

Demography-Based models build recommendations using user attribute values. Such a models are useful especially in cold-start user situations, where there is not enough interactions for the given u in the corresponding $\mathbf{r}_{u,*}$ row of the rating matrix. Usefulness of item i to user u is estimated based on user attribute values $a^U(u)$.

There are multiple possibilities of doing so. A traditional approach is to construct similarity function on top of user attributes, $sim\colon U^2 \to \mathbb{R}$, or, more precisely

$$sim\colon \left(\mathrm{dom}\left(A_1^U\right) \times \ldots \times \mathrm{dom}\left(A_r^U\right)\right)^2 \to \mathbb{R} \tag{2.14}$$

in compliance with 2.1.1.

More sophisticated approaches of building the models using deep learning methods were proposed in [20, 28]. These methods combine rating matrix data together with item and user attribute values when training a neural network, capturing possible non-linear dependencies between them.

2.3.5 Hybrid Systems

Besides pure Collaborative Filtering or Content-Based systems, many modern RSs can be characterized as hybrid. Hybrid recommender systems combine both approaches, typically using model ensembles (voting or chaining on top of multiple CF and CB models), or by using model consuming mixed types of data. We will give example of such model in Sec. 2.4.6.1, but others include e.g. [71], [29], [106], or even [52] (bachelor thesis supervised by the author of this thesis).

2.4 Recommendation Models

Vast majority of algorithms in Recommender Systems build on the idea of **scoring** individual items $i \in I$ in the current context, given typically by source user $u \in U$. This leads to introduction of the **score function**

$$score\colon U \times I \to \mathbb{R} \tag{2.15}$$

shared in some form by most of the models. The score function may be thought as expressing the *relevance* or the *usefulness* of item i to user u.

In different recommendation tasks, the score may be used for different purposes:

- estimate the rating in Rating Prediction, possibly by directly returning $\text{model}(u, i) = score(u, i)$ if properly normalized,
- finding relevant items in Binary Classification tasks, possibly by applying threshold $\theta \in \mathbb{R}$ to individual scored items to classify items as either irrelevant (0) or relevant (1) using formula

$$\text{model}(u, i) = \begin{cases} 0 & \text{if } score(u, i) < \theta \\ 1 & \text{if } score(u, i) \geq \theta \end{cases} \tag{2.16}$$

- determining the set of N best items in Top-N recommendation:

$$\begin{aligned} & \text{model}(u) = \{i_1, \dots i_N\} \\ \text{w.r.t.} \quad & \forall i_k \in \text{model}(u)\colon \; i_k \in I \wedge \\ & \qquad\qquad |\{j \in I \setminus \{i_k\} \mid score(u, j) > score(u, i_k)\}| < N \end{aligned} \tag{2.17}$$

- determining the ordering of items in ranking tasks:

$$\begin{aligned} \text{model}(u) &= \left(i_1, \dots, i_{|I|}\right) \\ \text{w.r.t. } \forall k \in \{1, \dots, |I|-1\} &: score(u, i_k) \geq score(u, i_{k+1}) \end{aligned} \tag{2.18}$$

Below in this section, some of the most important recommendation models are introduced. Keeping in mind the above mentioned, the models will be described within a uniform framework based on scoring, introducing different score functions for individual models.

2.4.1 Popularity Models

Popularity-based approaches are of the simplest possible mechanisms to generate likely useful recommendations to users. The models do not take possible preferences of the source user u into account at all, so we can say the recommendations are *not personalized* as it typically holds $score(u, i) = score(v, i)\ \forall u \in U, v \in U, i \in I$.

Given rating matrix $\mathbf{R}$, the score can be easily computed as

$$score(u, i) = \sum_{\substack{v \in U \setminus \{u\} \\ r_{v,i} \neq ?}} r_{v,i} \tag{2.19}$$

Using different filtering functions φ (as introduced in 2.1.4) to produce the rating matrix $\mathbf{R}$ may give different behavior for the resulting model. For example, taking only interactions from the last month or less can be used to get items that were popular in the recent period. Similarly, we can only take certain type of interactions, such as purchases to get the most commonly purchased items. Moreover, by using different Γ function (again as in 2.1.4), such as putting prices of the purchased items into the purchase-based rating matrix, we will get recommendations of items that overall generated the highest revenue.

Being very simple to implement yet still actually considering the rating data, popularity-based models can be used as a good baseline when evaluating more sophisticated approaches.

2.4.2 User-Based k-Nearest Neighbors

User-Based k-Nearest Neighbors model (or simply *UserKNN* for the rest of this thesis) is one the best-known approaches to generating personalized recommendations. In the simplest form, the model supposes existence of **user similarity** function $sim: U^2 \to \mathbb{R}$, together with parameter $k \in \mathbb{N}$. The function of the model is easiest to describe in two steps:

1. Using the *sim* function, find the set $\mathrm{NN}_k(u)$ of k users who are the most similar to source user $u \in U$:

$$\begin{aligned} & \mathrm{NN}_k(u) = \{v_1, \ldots, v_k\} \\ \text{w.r.t.} \quad & \forall v \in \mathrm{NN}_k(u): \ v \in U \setminus \{u\} \wedge \\ & \qquad |w \in U \setminus \{u\} \mid sim(u, w) > sim(u, v)| < k \end{aligned} \tag{2.20}$$

2. Use rating matrix values of the users from $\mathrm{NN}_k(u)$ to score the items. This is done either in weighted on unweighted way. For the Rating Prediction task, it is common to use average of the item's ratings made by users in the neighborhood:

$$score(u,i) = \begin{cases} \dfrac{\sum\limits_{\substack{v \in \mathrm{NN}_k(u) \\ r_{v,i} \neq ?}} r_{v,i}}{|\{v \in \mathrm{NN}_k(u) \mid r_{v,i} \neq ?\}|} & \text{if } \exists v \in \mathrm{NN}_k(u) \colon r_{v,i} \neq ? \\ 0 & \text{otherwise} \end{cases} \tag{2.21}$$

Alternatively to the above unweighted version, it is more common to use weighted averages when computing the score:

$$score(u,i) = \begin{cases} \dfrac{\sum\limits_{\substack{v \in \mathrm{NN}_k(u) \\ r_{v,i} \neq ?}} sim(u,v) \cdot r_{v,i}}{\sum\limits_{\substack{v \in \mathrm{NN}_k(u) \\ r_{v,i} \neq ?}} sim(u,v)} & \text{if } \exists v \in \mathrm{NN}_k(u) \colon r_{v,i} \neq ? \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

While for Rating Prediction it makes sense to normalize the score using division by sum of user similarities in the neighborhood, for other tasks, such as the Top-N recommendation, such normalization doesn't make much sense. Specifically, normalization brings loss of certainty. When an item has high score computed on a set of neighboring users, it makes difference if we know that the set of neighbors was big: the item is then a very good candidate for recommendation. Otherwise, the high score may come simply from noise, such as if there was only one user in the neighborhood who produced the high rating for that item.

Furthermore, normalization only makes sense of explicit rating matrices. In implicit rating matrices, most of the observed values are the same, all of them meaning somehow positive preference. In the simplest case, all the observed values may be equal to 1, and hence the above formulas would result in constant $score(u,i) = 1$. Actually, it is the number of neighboring users which interests us in such case.

To address these issues, **non-normalized neighborhood** has been introduced in [23]. The formula is simply:

$$score(u,i) = \begin{cases} \sum\limits_{\substack{v \in \mathrm{NN}_k(u) \\ r_{v,i} \neq ?}} sim(u,v) \cdot r_{v,i} & \text{if } \exists v \in \mathrm{NN}_k(u) \colon r_{v,i} \neq ? \\ 0 & \text{otherwise} \end{cases} \tag{2.23}$$

From now on, whenever referring to the *UserKNN* algorithm, we always mean the version with non-normalized neighborhood.

Besides the ways of computing the score, it is also the *sim* function which is absolutely crucial for the behavior of the algorithm.

In most literature, similarity computed from the rating matrix is considered. The two most popular measures are:

- Cosine similarity:

$$sim(u,v) = \frac{\mathbf{r}_{u,*} \cdot \mathbf{r}_{v,*}^T}{\|\mathbf{r}_{u,*}\| \cdot \|\mathbf{r}_{v,*}\|} = \frac{\sum\limits_{\substack{i \in I \\ r_{u,i} \neq ? \\ r_{v,i} \neq ?}} r_{u,i} \cdot r_{v,i}}{\sqrt{\sum\limits_{\substack{i \in I \\ r_{u,i} \neq ?}} r_{u,i}^2} \cdot \sqrt{\sum\limits_{\substack{i \in I \\ r_{v,i} \neq ?}} r_{v,i}^2}} \tag{2.24}$$

  In special case of binary implicit rating matrices, where all observed values equal to 1, that is $\forall i \in I, u \in U: r_{u,i} \in \{?, 1\}$, rows of the rating matrix can be thought as sets ($\hat{\mathbf{r}_u} = \{i \in I \mid r_{u,i} = 1\}$), simplifying the above formula to Ochiai similarity coefficient:

$$sim(u,v) = \frac{|\hat{\mathbf{r}_u} \cap \hat{\mathbf{r}_v}|}{|\hat{\mathbf{r}_u}| \cdot |\hat{\mathbf{r}_v}|} \tag{2.25}$$

- Pearson correlation coefficient:

$$sim(u,v) = \frac{\sum\limits_{\substack{i \in I \\ r_{u,i} \neq ? \\ r_{v,i} \neq ?}} (r_{u,i} - \overline{\mathbf{r}_{u,*}}) \cdot (r_{v,i} - \overline{\mathbf{r}_{v,*}})}{\sqrt{\sum\limits_{\substack{i \in I \\ r_{u,i} \neq ? \\ r_{v,i} \neq ?}} (r_{u,i} - \overline{\mathbf{r}_{u,*}})^2} \cdot \sqrt{\sum\limits_{\substack{i \in I \\ r_{u,i} \neq ? \\ r_{v,i} \neq ?}} (r_{v,i} - \overline{\mathbf{r}_{v,*}})^2}} \tag{2.26}$$

  Using Pearson similarity coefficient makes sense again mostly on explicit rating matrices with enough and meaningful variance across the ratings.

By properly changing the *sim* function, UserKNN can be very straightforwardly used for Demographic Recommendation as well [16]. Providing similarity function on top of user attribute values, as defined in Eq. 2.14, is sufficient for that. The resulting model can be used for recommendations to cold-start users, for who there aren't enough interactions yet (or no interactions at all, $\forall i \in I: r_{u,i} = ?$), but there are user attribute values available. They may be encode some apriori knowledge, such as data from a questionnaire filled by the user when registering to the system, or information about the source website that the user

came from. Similar users are determined using the attributes, and recommendations are generated from the rating matrix rows of those who already performed some interactions. This can be roughly understood are recommending the most popular items across the cluster of users having similar attributes.

2.4.3 Item-Based k-Nearest Neighbors

Item-Based k-Nearest Neighbors model (or simply *ItemKNN* through the rest of this thesis) has been introduced in [93] as a method of processing the rating matrix in a way somehow complementary to UserKNN. Instead of measuring similarity between users, the *sim* function focuses on **item similarity**:

$$sim\colon I^2 \to \mathbb{R} \tag{2.27}$$

While in UserKnn the *sim* is used to build a single set of k neighbors $\mathrm{NN}_k(u)$ that are most similar to u, ItemKNN builds n different neighborhoods, one for each observed item in the source user's rating vector ($n = |\{i \in I \mid r_{u,i} \neq ?\}|$). Items and their similarities from individual neighborhoods are then combined together using weights given by ratings of the corresponding source items in $\mathbf{r}_{u,*}$.

The two steps to describe how the model works are:

1. Using the *sim* function, for each item $i \in I$ such that $r_{u,i} \neq ?$, find a set $\mathrm{NN}_k(i)$ of k items that are the most similar to i:

$$\begin{aligned} \forall i \in \{i' \in I \mid r_{u,i'} \neq ?\}\colon \quad & \mathrm{NN}_k(i) = \{j_1, \ldots, j_k\} \\ \text{w.r.t.} \quad & \forall j \in \mathrm{NN}_k(i)\colon \quad j \in I \setminus \{i\} \wedge \\ & \qquad |\ell \in I \setminus \{i\} \mid sim(i,\ell) > sim(i,j)| < k \end{aligned} \tag{2.28}$$

2. Compute the score for item i from neighborhoods in which it is present. In the simplest case, no similarity weighting is performed and only the corresponding ratings are averaged:

$$score(u,i) = \begin{cases} \dfrac{\sum\limits_{\substack{j \in I \\ r_{u,j} \neq ? \\ i \in \mathrm{NN}_k(j)}} r_{u,j}}{|\{j \in I \mid r_{u,j} \neq ? \wedge i \in \mathrm{NN}_k(j)\}|} & \text{if } \exists j \in I\colon r_{u,j} \neq ? \wedge i \in \mathrm{NN}_k(j) \\ 0 & \text{otherwise} \end{cases} \tag{2.29}$$

The above formula works mainly for explicit rating matrices: simply average the observed ratings made by the source user u on the items to which i is similar (that

is, it lies in their neighborhoods). The assumption is that the user would rate similar items similarly.

Similarly to UserKNN (Eq. 2.22), the *score* function can be further adjusted to also take the similarities into account, resulting in **similarity-weighted average**:

$$score(u,i) = \begin{cases} \dfrac{\sum\limits_{\substack{j \in I \\ r_{u,j} \neq ? \\ i \in \mathrm{NN}_k(j)}} sim(i,j) \cdot r_{u,j}}{\sum\limits_{\substack{j \in I \\ r_{u,j} \neq ? \\ i \in \mathrm{NN}_k(j)}} sim(i,j)} & \text{if } \exists j \in I : r_{u,j} \neq ? \wedge i \in \mathrm{NN}_k(j) \\ 0 & \text{otherwise} \end{cases} \tag{2.30}$$

For implicit rating matrices, **non-normalized neighborhood** introduced in [23] can be applied in a way similar to UserKNN in Eq. 2.23:

$$score(u,i) = \begin{cases} \sum\limits_{\substack{j \in I \\ r_{u,j} \neq ? \\ i \in \mathrm{NN}_k(j)}} sim(i,j) \cdot r_{u,j} & \text{if } \exists j \in I : r_{u,j} \neq ? \wedge i \in \mathrm{NN}_k(j) \\ 0 & \text{otherwise} \end{cases} \tag{2.31}$$

While the UserKNN Eq. 2.23 preserves the information on number of users $v \in U$ who contributed on scoring i by being similar to source user u and having $r_{v,i} \neq ?$ in their rating matrix row, in ItemKNN, Eq. 2.31 preserves the information on number of items $j \in I$ which contributed on scoring i by being similar to i and having $r_{u,j} \neq ?$ in their rating matrix column.

2.4.3.1 Interaction Similarity

The similarity functions $sim: I^2 \to \mathbb{R}$ proposed in the original paper [93] are based on the rows of the rating matrix: row-oriented cosine similarity and Pearson correlation similarity. Since the formulas are symmetrical to Eqs. 2.24, 2.25, and 2.26, we will not repeat their transposed versions here. All the formulas apply also for ItemKNN just by replacing rows with columns and measuring similarity between $\mathbf{r}_{*,i}$ and $\mathbf{r}_{*,j}$ instead of $\mathbf{r}_{u,*}$ and $\mathbf{r}_{v,*}$.

2.4.3.2 Attribute Similarity

The set of possible similarity functions goes beyond the rating matrix. **Item attributes** may be considered instead of ratings, resulting in Content-Based recommendation model which is suitable for cold-start items which haven't collected enough interactions yet.

Given set of item attributes $\{A_1^I, \ldots, A_r^I\}$ with function $a^I: I \to \mathrm{dom}\left(A_1^I\right) \times \ldots \times \mathrm{dom}\left(A_r^I\right)$ where $\mathrm{dom}\left(A_j^I\right)$ may be $\mathbb{R}$, $\mathbb{Z}$, or $\mathbb{N}$ for numeric attributes, Σ^* for textual attributes, 2^{Σ^*} for set attributes (such as names of categories which the item belongs to), $\{0, 1\}$ for booleans, or even more complex structures such as images, the natural idea is to put

$$sim(i, j) = attrsim(a^I(i), a^I(j)) \tag{2.32}$$

where $attrsim: \left(\mathrm{dom}\left(A_1^I\right) \times \ldots \times \mathrm{dom}\left(A_r^I\right)\right)^2$ is a function measuring similarity between two items based on the values of their attributes. In this thesis, besides rating similarity, we will also consider similarity of tokens.

Token similarity

Similarity function based on introduction of a set of binary features through *tokenization* as described in 2.1.2. Considering $tokenization: \mathrm{dom}\left(A_1^I\right) \times \ldots \times \mathrm{dom}\left(A_r^I\right) \to 2^{\Sigma^+}$, the similarity between two items can be computed using cosine similarity implemented as Jaccard similarity coefficient

$$attrsim(a_i, a_j) = \frac{|tokenization(a_i) \cap tokenization(a_j)|}{|tokenization(a_i) \cup tokenization(a_j)|} \tag{2.33}$$

Image and Embedding similarity

In a bachelor thesis supervised by the author of this thesis [79] (in Czech), several approaches have been proposed to provide the ItemKnn with *sim* function able to measure item similarity based on **images**. Different methods have been studied, including the color histogram, ORB [86], and pre-trained convolutional neural network [98], leading to very interesting results. See Figs. 2.1, 2.2, and 2.3.

By taking activation values from a selected fully connected layer in the convolutional network, image attribute data (as defined in Sec. 2.1.1.4) can be transformed into vectors in $\mathbb{R}^n$, called the **embeddings** in general. Denoting $embedding\,(a_i) \in \mathbb{R}^n$ as the embedding of attribute values $a_i = a^I(i)$ of item $i \in I$, one can use e.g. the cosine similarity between embeddings as the Eq. 2.33 attribute similarity function:

$$attrsim(a_i, a_j) = \frac{embedding(a_i)^T \cdot embedding(a_j)}{\|embedding(a_i)\| \cdot \|embedding(a_j)\|} \tag{2.34}$$

This applies as well to embeddings of types other than image, such as the *word2vec* [72] for text attributes, *Soundnet* [5] for audio attributes, etc.

In another bachelor thesis also supervised by the author of this thesis [52], a feedforward neural network estimating the rating (interaction) similarity (as in Sec. 2.4.3.1) based on the provided attributes, is proposed. The idea is that rating similarity typically leads to the best results on mainstream items, but cannot be used on cold-start items having not enough interactions collected yet. Henceforth, a training dataset is introduced, containing pairs of

Figure 2.1: Two sets of nearest neighbors (to the top-left item) using color histogram as the *sim* function in ItemKnn [79].

Figure 2.2: Two sets of nearest neighbors (to the top-left item) using ORB [86] as the *sim* function in ItemKnn [79].

item attribute values $\left(a^I(i), a^I(j)\right)$ (see Sec. 2.1.1) and the associated rating (interaction) similarity $sim_{\mathbf{R}}(i,j)$ between the two items. That is:

$$\mathcal{T} = \left\{ \left(\left(\left(a^I\left(i_1\right), a^I\left(j_1\right)\right), sim_{\mathbf{R}}\left(i_1, j_1\right)\right), \ldots, \left(a^I\left(i_\ell\right), a^I\left(j_\ell\right)\right), sim_{\mathbf{R}}\left(i_\ell, j_\ell\right)\right)\right\} \tag{2.35}$$

using notion of training set systematically introduced later in this thesis in Sec. 2.5.1. Based on attribute values and known interaction similarities of pairs of items where $sim_{\mathbf{R}}(i_\ell, j_\ell)$ is known with high confidence, because these items have enough interactions, the neural network is trained. Later, for previously unseen pairs $(i_k, j_k) \in I^2$, the network predicts $\widehat{sim_{\mathbf{R}}}\left(i_k, j_k\right)$ just based on the attribute values $\left(a^I\left(i_k\right), a^I\left(j_k\right)\right)$, solving the cold start problem on some datasets.

Figure 2.3: Two sets of nearest neighbors (to the top-left item) using pre-trained convolutional network [98] as the *sim* function in ItemKnn [79].

2.4.4 Association Rules

The problem of association rules mining was stated in [2] and further extended in [3], both of which are of the most cited articles in Computer Science. Association Rules (or *ARs* for short) are simple probabilistic statements about co-occurrences of events in data. They have useful applications in many different areas.

Because ARs are not as popular as other approaches (neighborhood-based models, matrix factorization, deep learning), the following survey and unified framework for ARs-based recommendation is one of the contributions of this thesis.

In their basic form, ARs require discrete data on the input. Rather than rating vectors from $(\mathbb{R} \cup \{?\})^{|I|}$, users are viewed as sets of items from 2^I. We will denote user u represented in this way as $T(u)$ and will refer to it as to **transaction** of user u. Having users represented as transactions, each item $i \in I$ is considered either strictly **relevant** ($i \in T(u)$) or strictly **irrelevant** ($i \notin T(u)$) for a given user $u \in U$.

For each user $u \in U$, the transaction $T(u)$ can be easily computed from the rating matrix using threshold $\theta \in \mathbb{R}$:

$$T(u) = \{i \in I \mid r_{u,i} \neq ? \wedge r_{u,i} \geq \theta\} \tag{2.36}$$

2.4.4.1 Mining Frequent Itemsets

The rules themselves are statements in form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$, $X \neq \emptyset$, $Y \neq \emptyset$, and $X \cap Y = \emptyset$. Whenever we use the $X \Rightarrow Y$ notation in this thesis, we assume these conditions are met for X and Y. In its traditional form, the learning algorithm which produces the model has the **minimal support** parameter $s_{\min} \in [0, 1]$.

Let us denote the **support** function $supp: 2^I \to [0,1]$ that, for given set $A \subseteq I$, computes the portion of users having A as subset of their transaction:

$$supp(A) = \frac{|\{u \in U \mid A \subseteq T(u)\}|}{|U|} \tag{2.37}$$

Given the set of user transactions and the $s_{\min}$ parameter, we can generate the set $\mathcal{R}$ of basic association rules holding the minimal support condition as:

$$\mathcal{R} = \{X \Rightarrow Y \mid supp\,(X \cup Y) \geq s_{\min}\} \tag{2.38}$$

Different algorithms can be used to solve the task of mining the $\mathcal{R}$ set. One of the most popular is the APRIORI algorithm also proposed in [3]. The algorithm starts with searching for support of elementary itemsets, consisting of single item only. After testing these elementary itemsets, we obtain $\mathcal{S}^1$, which is set of frequent itemsets S of cardinality $|S| = 1$.

Given set $\mathcal{S}^k = (S_1^k, S_2^k, \ldots, S_r^k)$ of frequent itemsets of cardinality $k \in \mathbb{N}$, we can efficiently construct set of candidate itemsets $\mathcal{C}^{k+1} = (C_1^{k+1}, C_2^{k+1}, \ldots, C_s^{k+1})$ of cardinality $k+1$. This means that itemsets can be generated in levels. See the pseudocode for APRIORI in Alg. 1.

Algorithm 1: APRIORI

input : Set of items I, set of user transactions $\{T(u) \mid u \in U\}$, minimal support $s_{\min} \in [0,1]$

output: Set of frequent itemsets

$\mathcal{C}^1 \leftarrow \{\{i\} \mid i \in I\}$
$k \leftarrow 1$
repeat
  $\mathcal{S}^k \leftarrow \{C \in \mathcal{C}^k \mid supp(C) \geq s_{min}\}$
  $\mathcal{C}^{k+1} \leftarrow \left\{ S_A^k \cup S_B^k \;\middle|\; S_A^k, S_B^k \in \mathcal{S}^k, |S_A^k \cup S_B^k| = k+1 \right\}$
  $k \leftarrow k+1$
until $\mathcal{C}^k = \emptyset$
return $\bigcup_{j=1}^{k-1} \mathcal{S}^j$

Naïve implementation of Alg. 1 has time complexity $O\left(\left|\mathcal{S}^k\right|^2\right)$ of building the $(k+1)$-th level of itemsets. Several techniques may be utilized to avoid duplicities. Exploiting the fact that the set of items I is totally ordered (e.g. using total order on unique identifiers), frequent itemsets at level k can be seen as sequences. Considering two sequences $(i_1^A, i_2^A, \ldots, i_{k-1}^A, i_k^A)$ and $(i_1^B, i_2^B, \ldots, i_{k-1}^B, i_k^B)$, we can introduce a rule that these two sequences can only be merged if they overlap at $k-1$ elements:

$$(i_2^A = i_1^B) \wedge (i_3^A = i_2^B) \wedge \ldots \wedge (i_{k-1}^A = i_{k-2}^B) \wedge (i_k^A = i_{k-1}^B). \tag{2.39}$$

This will avoid identical itemsets being generated more than once, preserving completeness of the algorithm.

Besides APRIORI, other algorithms have been introduced, addressing different types performance limitations in some cases. The ECLAT [110] and FP-Growth [41] are the most prominent.

Provided the resulting set of frequent itemsets $F = \bigcup_{j=1}^{k-1} \mathcal{S}^j$, individual association rules can be generated by splitting them to left-hand and right-hand sides in brute-force manner, simply assuming that they are small enough to be able to do so:

$$\mathcal{R} \leftarrow \{X \Rightarrow Y \mid X \cup Y \in F\} \tag{2.40}$$

2.4.4.2 Rule Quality Measures

Discovered set $\mathcal{R}$ of basic association rules is based on frequent itemsets only, ensuring for each rule $X \Rightarrow Y$ that X and Y occur together in at least $\lceil s_{\min} \cdot |U| \rceil$ user transactions. It does not, however, take into account the direction of the rules as it holds $(X \Rightarrow Y) \in \mathcal{R} \iff (Y \Rightarrow X) \in \mathcal{R}$. Therefore, different **rule quality measures** have been invented to further rate the rules. Rule quality measure is simply function $q: \mathcal{R} \rightarrow \mathbb{R}$. A comprehensive list of different versions of q is studied and evaluated in [6]. Let us briefly introduce the most relevant ones for this thesis.

Confidence

The very first measure, proposed by the authors of the APRIORI algorithm in [3]. It measures how often the rules are (not) violated on the set of all user transactions in terms of satisfied left-hand side and unsatisfied right-hand side:

$$\mathit{conf}\,(X \Rightarrow Y) = \frac{\mathit{supp}\,(X \cup Y)}{\mathit{supp}(X)} \tag{2.41}$$

Lift

This measure addresses the fact that some rules may have high confidence only by chance, even through X and Y are statistically independent. Hence it measures how far from statistical independence X and Y are, using the formula [6]:

$$\mathit{lift}\,(X \Rightarrow Y) = \frac{\mathit{conf}\,(X \Rightarrow Y)}{\mathit{supp}(Y)} \tag{2.42}$$

Conviction

Reflects the weakness of *lift*, for which it holds $\mathit{lift}\,(X \Rightarrow Y) = \mathit{lift}\,(Y \Rightarrow X)$, although association rules are of unidirectional nature. It is measured as:

$$\mathit{conv}\,(X \Rightarrow Y) = \frac{1 - \mathit{supp}(Y)}{1 - \mathit{conf}\,(X \Rightarrow Y)} \tag{2.43}$$

2.4.4.3 Rule-Based Recommendation

Given set $\mathcal{R}$ of discovered association rules, and rule quality function q, it is possible to construct the *score*: $U \times I \to \mathbb{R}$ function same as for other recommendation models. The basic idea lies in picking transaction $T(u)$ of the source user u, using the rules $(X \Rightarrow Y)$ where $X \subseteq T(u)$ and scored item $i \in Y$, and taking into account their quality $q\,(X \Rightarrow Y)$.

In [91], Top-N Recommendation task is considered, and the following simple algorithm has been proposed: When generating recommendations for user u, we collect all the rules $X \Rightarrow Y$ such that $X \subseteq T(u)$. We then sort the rules in descending order by q and recommend the Top-N items from the right-hand sides of the rules at the top of the list. If an item is predicted by multiple rules, we only use the rule with the highest value of the measure function. The rule-quality measure proposed in [91] is *confidence*. A similar, yet more detailed algorithm, which additionally uses support of the rule and the left-hand side cardinality to break ties, is proposed in [61]. In this paper, we call this approach the **best-rule** method. It can be generalized and expressed as the *score* function as follows:

$$score(u,i) = \begin{cases} \max\limits_{\substack{(X \Rightarrow Y) \in \mathcal{R} \\ T(u) \subseteq X \\ i \in Y}} \left(q\,(X \Rightarrow Y)\right) & \text{if } \exists\,(X \Rightarrow Y) \in \mathcal{R}\colon X \subseteq T(u) \wedge i \in Y \\ 0 & \text{otherwise} \end{cases} \tag{2.44}$$

The appropriateness of the *confidence* measure is addressed in [6], where many other measures, such as the *lift* and the *conviction*, are compared by performance in classification tasks. It [6], it is also mentioned that there are other possibilities than the *best-rule* method. **Weighted voting** method as proposed in [54] is discussed as one of the options. Generalizing as the *score* function, the idea lies simply in replacing the maximum with the sum, allowing multiple rules to vote and contribute to the score of the item:

$$score(u,i) = \begin{cases} \sum\limits_{\substack{(X \Rightarrow Y) \in \mathcal{R} \\ T(u) \subseteq X \\ i \in Y}} q\,(X \Rightarrow Y) & \text{if } \exists\,(X \Rightarrow Y) \in \mathcal{R}\colon X \subseteq T(u) \wedge i \in Y \\ 0 & \text{otherwise} \end{cases} \tag{2.45}$$

It is important to keep in mind that the behavior of Eqs. 2.44 and 2.45 strongly depends on the $s_{\min}$ parameter which controls the granularity of rules in $\mathcal{R}$ in Eq. 2.38, threshold parameter θ for generating the transactions in Eq. 2.36, and the choice of rule quality measure q, for which Eqs. 2.41, 2.42, and 2.43 are some of the candidates.

Besides the mentioned, several other approaches have been proposed to use the Association Rules for recommendation. An approach to CF using ARs, together with an algorithm to mine ARs adaptively for each user, is presented in [65]. A framework for personalized recommendation using ARs and Sequential Patterns has been proposed in [62]. In [90], the robustness of ARs is discussed regarding resistance of the RS to attacks. In [73], ARs are presented as an efficient tool to handle extreme sparsity of data matrices

in context of web-content recommendation. Moreover, [59] shows that ARs are able to handle cold-start problems, i.e. situations where new items or users appear in the database. Another examples of rule-based recommender systems include [114, 112]. In the thesis, however, we will focus on the *best-rule* and *weighted voting* methods as defined above.

2.4.5 Matrix Factorization

Matrix Factorization (or just *MF* for short) techniques belong to group of latent factor models in Collaborative Filtering. Other examples of such models include Latent Dirichlet Allocation (LDA) [46], which is also useful for example for processing text attributes in Content-Based Recommendation [14], or Restricted Boltzmann Machines (RBM) in Collaborative Filtering [89]. The aim of latent factor models is to uncover latent features that would explain what is present in the provided data.

MF gained its popularity during the Netflix Prize Challenge in 2006 where they were used by the winning team BellKor (Robert Bell and Yehuda Koren) as the predominant model in the final model ensemble.[10]. However, the method was introduced several years earlier in [92] and later extended in [91] (Badrul Sarwar at GroupLens Research Group), based on technique known from linear algebra as Singular Value Decomposition (SVD). In the Netflix Challenge datasets, same as in the MovieLens datasets used by GroupLens, explicit rating matrices were used with model evaluations as the performance on Rating Prediction tasks. This impacted the applicability of the researched methods: Most of them were originally designed to fit this scenario.

A good and systematic introduction to MF models is [56]. Given the rating matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{U \times I}$, we want to infer user and item representations of latent features in $\mathbb{R}^f$ with $f \in \mathbb{N}$: $f \ll |U| \wedge f \ll |I|$. Put together, these features form the user features matrix $\mathbf{P} \in \mathbb{R}^{f \times |U|}$ and the item features matrix $\mathbf{Q} \in \mathbb{R}^{f \times |I|}$. Using these two matrices, we are trying to decompose the rating matrix $\mathbf{R}$ into a lower-rank representation and make *score* prediction as a dot product of the corresponding user and item feature vectors [56]:

$$score(u, i) = \mathbf{q}_{*,i}^T \cdot \mathbf{p}_{*,u} \tag{2.46}$$

While the *score* computation is this trivial for given matrices $\mathbf{P}$ and $\mathbf{Q}$, the difficult part is computing these from $\mathbf{R}$. Because [56] comes from the Netflix Prize, where explicit ratings dataset was provided, it assumes that $\mathbf{R}$ in an explicit rating matrix. It states search for $\mathbf{P}$ and $\mathbf{Q}$ as optimization problem:

$$\min_{\substack{\mathbf{P} \in \mathbb{R}^{f \times |U|} \\ \mathbf{Q} \in \mathbb{R}^{f \times |I|}}} \sum_{\substack{u \in U \\ i \in I \\ r_{u,i} \neq ?}} \left(r_{u,i} - \mathbf{q}_{*,i}^T \cdot \mathbf{p}_{*,u} \right)^2 + \lambda \left(\|\mathbf{q}_{*,i}\|^2 + \|\mathbf{p}_{*,u}\|^2 \right) \tag{2.47}$$

The model trains through optimizing squared error on the observed ratings. Regularization in form of ridge regression is used to avoid overfitting with λ as a hyperparameter.

Please note once again that this only make sense for explicit rating matrices. For implicit rating matrices, for similar reasons that led to introduction on non-normalized neighborhood in Eq. 2.23 in neighborhood-based models: In the simplest case of implicit rating matrices $\mathbf{R} \in (?, 1)^{|U|\times|I|}$, there exists trivial solutions such as $\forall j \in \{1, \ldots, f\}$, $\forall u \in U$, $\forall i \in I$: $p_{u,j} = q_{i,j} = \sqrt{\frac{1}{f}}$ or even $\forall u \in U$, $\forall i \in I$: $p_{u,1} = q_{i,1} = 1 \wedge \forall j \in \{2, \ldots, f\}$: $p_{u,j} = q_{i,j} = 0$, neither of which depends on data in $\mathbf{R}$ nor brings any value.

Multiple approaches exist to solve the Eq. 2.47 optimization problem. Some of the most popular are:

- Stochastic Gradient Descent (SGD),
- Alternating Least Squares (ALS),
- Conjugate Gradients (CG).

SGD is one of the simplest yet powerful enough methods. For each observed rating $r_{u,i} \neq ?$, we can compute the corresponding prediction $\mathbf{q}_i^T \cdot \mathbf{p}_u$, compare it to true value of $r_{u,i}$, and modify $\mathbf{p}_u$ and $\mathbf{q}_i$ in a direction opposite to the Eq. 2.47 error gradient as shown in Alg. 2.

Algorithm 2: Solving Matrix Factorization Optimization Problem Using SGD [56, 30]

input : Rating matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{|U|\times|I|}$, number of factors $f \in \mathbb{N}$, regularization $\lambda \in \mathbb{R}_0^+$, learning rate $\gamma \in \mathbb{R}_0^+$, number of steps $n \in \mathbb{N}$
output: Matrices $\mathbf{P} \in \mathbb{R}^{f\times|U|}$ and $\mathbf{Q} \in \mathbb{R}^{f\times|I|}$ minimizing Eq. 2.47

$\mathbf{P} \leftarrow$ init_matrix$(|U|, f)$
$\mathbf{Q} \leftarrow$ init_matrix$(|I|, f)$
for $\ell \leftarrow 1$ **to** n **do**
  for $(u, i) \in U \times I$ **do**
    if $r_{u,i} \neq ?$ **then**
      $e_{u,i} \leftarrow r_{u,i} - \mathbf{q}_{*,i}^T \cdot \mathbf{p}_{*,u}$
      $\mathbf{q}_{*,i} \leftarrow \mathbf{q}_i + \gamma\,(e_{u,i} \cdot \mathbf{p}_{*,u} - \lambda \cdot \mathbf{q}_{*,i})$
      $\mathbf{p}_{*,u} \leftarrow \mathbf{p}_u + \gamma\,(e_{u,i} \cdot \mathbf{q}_{*,i} - \lambda \cdot \mathbf{p}_{*,u})$
return $(\mathbf{P}, \mathbf{Q})$

This version of SGD is sequential. Efficient implementation builds a set of observed ratings $X = \{(u, i, x) \mid u \in U \wedge i \in I \wedge x = r_{u,i} \wedge r_{u,i} \neq ?\}$ and then iterates n times through this set in sequential manner. This cannot be naively parallelized, because while updating $\mathbf{q}_i$ and $\mathbf{p}_u$ for some $(u, i, r_{u,i}) \in X$, the whole subset $\{(u', i', r_{u',i'}) \mid u' = u \vee i' = i\} \subseteq X$ must be locked to prevent race conditions [64]. Several different approaches have been proposed to make parallelization possible. Some of the most popular include Delayed SGD

[57], Distributed SGD [34], Jellyfish [82], SimuParallelSGD [116], HOGWILD! [76], and FPSGD [115].

ALS is a different approach to minimize Eq. 2.47 by exploiting the structure of the problem. Because $\mathbf{P}$ and $\mathbf{Q}$ are two unknowns for which to optimize, the problem is not convex. However, if we fix one of them, the problem becomes quadratic and thus easy to solve optimally (find global optimum) using a standard least squares solver. Exploiting the fact that Eq. 2.47 corresponds to the standard form of ridge regression problems, for fixed $\mathbf{Q} \in \mathbb{R}^{f \times |I|}$, the solution for $\mathbf{p}_{*,u} \in \mathbb{R}^f$ can se found independently (and hence in parallel) for each user $u \in U$ as:

$$\mathbf{p}_{*,u} \leftarrow \left(\mathbf{Q} \cdot \mathbf{Q}^T + \lambda \cdot \mathbf{I}\right)^{-1} \cdot \mathbf{Q} \cdot \hat{\mathbf{r}_u}^T \tag{2.48}$$

where $\mathbf{I}$ is an $f \times f$ identity matrix and $\hat{\mathbf{r}_{*,u}} = \begin{cases} r_{u,i} & \text{if } r_{u,i} \neq ? \\ 0 & \text{otherwise} \end{cases} \forall i \in I.$

Symmetrically, for fixed $\mathbf{P} \in \mathbb{R}^{f \times |U|}$, solution for $\mathbf{q}_i \in \mathbb{R}^f$ can be found independently for each item $i \in I$ as:

$$\mathbf{q}_{*,i} \leftarrow \left(\mathbf{P} \cdot \mathbf{P}^T + \lambda \cdot \mathbf{I}\right)^{-1} \cdot \mathbf{P} \cdot \hat{\mathbf{r}_i}^T \tag{2.49}$$

where $\hat{\mathbf{r}_i} = \begin{cases} r_{u,i} & \text{if } r_{u,i} \neq ? \\ 0 & \text{otherwise} \end{cases} \forall u \in U$ is rating matrix column with $? \rightarrow 0$ replacements.

Learning is done in iterations where we alternate between fixing user factors while computing item factors, and fixing item factors while computing user factors.

Thanks to the fact that updates to, say, user factors, while keeping item factors fixed, are independent of each other, ALS can be efficiently parallelized in much easier and straightforward way than SGD.

MF models allow incorporating additional sources of information so one can create more sophisticated models.

In [56], it is proposed to add user and item bias predictors $\mathbf{b}^U \in \mathbb{R}^{|U|}$ and $\mathbf{b}^I \in \mathbb{R}^{|I|}$, together with the global bias $\mu \in \mathbb{R}$ to compute the *score* as

$$score(u, i) = \mu + b_u^U + b_i^I + \mathbf{q}_{*,i}^T \cdot \mathbf{p}_{*,u}, \tag{2.50}$$

extending the Eq. 2.47 loss function of the optimization problem to

$$\min_{\substack{\mathbf{P} \in \mathbb{R}^{f \times |U|} \\ \mathbf{Q} \in \mathbb{R}^{f \times |I|} \\ \mathbf{b}^U \in \mathbb{R}^{|U|} \\ \mathbf{b}^I \in \mathbb{R}^{|I|} \\ \mu \in \mathbb{R}}} \sum_{\substack{u \in U \\ i \in I \\ r_{u,i} \neq ?}} \left(r_{u,i} - \mu - b_i^I - b_u^U - \mathbf{q}_{*,i}^T \cdot \mathbf{p}_{*,u}\right)^2 + \lambda \left(\|\mathbf{q}_{*,i}\|^2 + \|\mathbf{p}_{*,u}\|^2 + {b_i^I}^2 + {b_u^U}^2\right) \tag{2.51}$$

Again, such optimization only makes sense for Rating Prediction tasks with explicit rating matrices.

In [55], an interesting method has been proposed to also take implicit rating data into account. This was an important shift in thinking towards implicit rating matrices, that are the most common in real-world datasets as discussed in 2.1.3. The authors basically

work with 2 different rating matrices, $\mathbf{R}^R$ containing purely explicit and $\mathbf{R}^N$ purely implicit rating data. The central idea of the paper lies in changing the *score* function to

$$score(u,i) = b_{u,i} + \sum_{\substack{j \in I \\ r^R_{u,j} \neq ?}} \left(r^R_{u,j} - b_{u,j}\right) \cdot w_{i,j} + \sum_{\substack{j \in I \\ r^N_{u,j} \neq ?}} c_{i,j} \tag{2.52}$$

and let $\mathbf{B} \in \mathbb{R}^{|U| \times |I|}$, $\mathbf{W} \in \mathbb{R}^{|I| \times |I|}$, and $\mathbf{C} \in \mathbb{R}^{|I| \times |I|}$ optimize through different loss function:

$$\begin{aligned} \min_{\substack{\mathbf{B} \in \mathbb{R}^{|U| \times |I|} \\ \mathbf{W} \in \mathbb{R}^{|I| \times |I|} \\ \mathbf{C} \in \mathbb{R}^{|I| \times |i|} \\ \mathbf{b}^U \in \mathbb{R}^{|U|} \\ \mathbf{b}^I \in \mathbb{R}^{|I|} \\ \mu \in \mathbb{R}}} \sum_{\substack{u \in U \\ i \in I \\ r^R_{u,i} \neq ?}} \Bigg(\Bigg(\Bigg(r^R_{u,i} - \mu - b^U_u - b^I_i - \sum_{\substack{j \in I \\ r^R_{u,j} \neq ?}} \left(r^R_{u,j} - b_{u,j}\right) \cdot w_{i,j} - \sum_{\substack{j \in I \\ r^N_{u,j} \neq ?}} c_{i,j} \Bigg)^2 \\ + \lambda \left({b^U_u}^2 + {b^I_i}^2 + \sum_{\substack{j \in I \\ r^R_{u,j} \neq ?}} w^2_{i,j} + \sum_{\substack{j \in I \\ r^N_{u,j} \neq ?}} c^2_{i,j} \right) \Bigg) \end{aligned} \tag{2.53}$$

while proposing set of update rules for SGD to do the optimization. The aim of the paper was still Rating Prediction on explicit rating matrices, and the role of implicit ratings was to help lower the prediction error.

In [47], a novel approach designed specifically to handle **implicit matrices** has been proposed, extending the scope of MF methods to areas other than Rating Prediction. In context of this thesis, [47] can be considered as a breakthrough in the MF field. The authors identify important characteristics of implicit rating matrices, such as the absence of negative ratings and inability to use trained models for Rating Prediction tasks.

The paper presented a novel way of dealing with implicit rating matrix: decomposition of the rating into a preference and a confidence value. The preference matrix $\mathbf{Y} \in \{0,1\}^{|U| \times |I|}$ is a binary matrix which is not sparse anymore, unobserved interactions are treated as zeros:

$$y_{u,i} = \begin{cases} 1 & \text{if } r_{u,i} \neq ? \\ 0 & \text{if } r_{u,i} = ? \end{cases} \tag{2.54}$$

The confidence matrix $\mathbf{C} \in \mathbb{R}^{+|U| \times |I|}$ assigned each element in $\mathbf{Y}$ a weight

$$c_{u,i} = 1 + \alpha \cdot \begin{cases} r_{u,i} & \text{if } r_{u,i} \neq ? \\ 0 & \text{if } r_{u,i} = ? \end{cases} \tag{2.55}$$

with empirically suggested quite high value of $\alpha = 40$.

As a result, the method takes $? \to 0$ replacements into account an handles them in the same way as they were observed, minimizing $\left|\mathbf{q}^T_{*,i} \cdot \mathbf{p}_{*,u}\right|$ for them, but with much lower weight that the observed ones. This reflects two facts:

1. There is uncertainty about unobserved ratings: missing rating does not necessarily mean zero preference, the user might be just unaware of the existence of given item.

2. The rating matrix is highly sparse, resulting in vast majority of 0s when $? \rightarrow 0$ replacement is applied.

The proposed loss function of the optimization problem is:

$$\min_{\substack{\mathbf{P}\in\mathbb{R}^{f\times U}\\ \mathbf{Q}\in\mathbb{R}^{f\times I}}} \sum_{\substack{u\in U\\ i\in I}} \left(c_{u,i} \cdot \left(y_{u,i} - \mathbf{q}_{*,i}^T \cdot \mathbf{p}_{*,u} \right)^2 \right) + \lambda \left(\sum_{u\in U} \|\mathbf{p}_{*,u}\|^2 + \sum_{i\in I} \|\mathbf{q}_{*,i}\|^2 \right) \tag{2.56}$$

This approach has been adopted in industry as a way of dealing with implicit feedback (e.g. Spotify [49], Quora [107]).

It is noteworthy that the optimization process takes into account the whole rating matrix, making it impossible to use certain techniques, such as the optimized ALS as described in [11]. The authors of [47], however, came with an optimized solution by exploiting the structure of the problem and the decomposition to be able to effectively compute the solution using only the observed ratings. Assuming fixed $\mathbf{Q} \in \mathbb{R}^{f\times|I|}$, the solution for $\mathbf{p}_u \in \mathbb{R}^f$ can be found independently for each $u \in U$ as:

$$\mathbf{p}_u = \left(\mathbf{Q} \cdot \mathbf{C}^u \cdot \mathbf{Q}^T + \lambda \mathbf{I} \right)^{-1} \cdot \mathbf{Q} \cdot \mathbf{C}^u \cdot \mathbf{y}_{u,*}^T \tag{2.57}$$

where $\mathbf{C}^u \in \mathbb{R}^{|I|\times|I|}$ is a diagonal matrix with $\forall i \in I: c_{i,i}^u = c_{u,i}$, and $\mathbf{I}$ is an $f \times f$ identity matrix.

Symmetrically, assuming fixed $\mathbf{P} \in \mathbb{R}^{f\times|U|}$, the solution for $\mathbf{q}_i \in \mathbb{R}^f$ can be computed using:

$$\mathbf{q}_i = \left(\mathbf{P} \cdot \mathbf{C}^i \cdot \mathbf{P}^T + \lambda \mathbf{I} \right)^{-1} \cdot \mathbf{P} \cdot \mathbf{C}^i \cdot \mathbf{y}_{*,i} \tag{2.58}$$

where $\mathbf{C}^u \in \mathbb{R}^{|U|\times|U|}$ is a diagonal matrix with $\forall u \in U: c_{u,u}^i = c_{u,i}$.

Another approach to dealing with missing ratings lies in imputation of non-zero values. Under the assumption that ratings are not missing at random (MNAR assumption), [100] proposes a MF model based on weighted-RMSE optimization problem on the same basis as above in [47].

The proposed *score* function in [100] is:

$$score(u,i) = r_m + \mathbf{q}_{*,i}^T \cdot \mathbf{p}_{*,u} \tag{2.59}$$

where $r_m \in \mathbb{R}$ is the imputed value.

The optimization problem states:

$$\min_{\substack{\mathbf{P}\in\mathbb{R}^{f\times|U|}\\ \mathbf{Q}\in\mathbb{R}^{f\times|I|}}} \sum_{\substack{u\in U\\ i\in I}} w_{u,i} \cdot \left(\left(\mathbf{r}_{u,i}^{\text{o\&i}} - r_m - \mathbf{q}_{*,i}^T \cdot \mathbf{p}_{*,u} \right)^2 + \lambda \cdot \sum_{\ell=0}^{f} \left(p_{\ell,u}^2 + q_{\ell,i}^2 \right) \right) \tag{2.60}$$

where $\mathbf{R}^{\text{o\&i}} \in \mathbb{R}^{|U|\times|I|}$ is the imputed rating matrix:

$$\forall u \in U, i \in I: r_{u,i}^{\text{o\&i}} = \begin{cases} r_{u,i} & \text{if } r_{u,i} \neq ? \\ r_m & \text{if } r_{u,i} = ? \end{cases} \tag{2.61}$$

and $\mathbf{W} \in \mathbb{R}^{|U|\times|I|}$ is the weight matrix:

$$\forall u \in U, i \in I: w_{u,i} = \begin{cases} w\left(r_{u,i}\right) & \text{if } r_{u,i} \neq ? \\ w_m & \text{if } r_{u,i} = ? \end{cases} \tag{2.62}$$

All the following:

- observed ratings weighting function $w \in \mathbb{R} \to \mathbb{R}$,
- imputed value $r_m \in \mathbb{R}$,
- weight of imputed values $w_m \in \mathbb{R}_0^+$,

are model hyperparameters proposed as subject to meta-optimization by means of cross-validation (see 2.5.1), with values like $w_m = 0.01$ and $r_m = 2$, and $w(r) = 1$ discovered in experimental part of [100]. Despite [100] focuses on explicit rating matrices, the method can be used for implicit matrices as well, being structurally similar to [47]. In [100], an efficient method to solve Eq. 2.60 optimization problem using ALS is proposed.

While in [100] are only single (chosen) values of $r_m \in \mathbb{R}$ and $w_m \in \mathbb{R}$ assigned to all the $r_{u,i} =?$, more complex methods have been proposed that require (u, i)-dependent weighting. In [101], item-dependent weighting based on overall item popularity in the rating matrix has been proposed (this is discussed in more detail below and later in this thesis).

There are many other possible weighting strategies, and the intuitions behind them may not seem obvious. User-oriented approach from [77] postulates that users with more observed ratings are more likely to dislike unobserved items, and hence proposes higher weights for missing ratings for such users. But completely different reasoning can be used as well: The more observed ratings (positive examples in implicit feedback), the higher possibility to also like the unknowns, and thus lower weight for the (negative) missing ratings for such user. This dichotomy is mentioned in [109], referring to [53] where just the opposite strategy compared to [77] has been used.

Most of the mentioned user-/item-dependent weighting strategies suffer from computational inefficiency as in naive implementation, they require computations on the full $|U|\times|I|$ imputed rating matrix and not only the observed ratings, size of which may easily go to trillions or quadrillions on real-world datasets. In [109], a novel way of using specific weights for the unknowns while keeping the computation efficient, i.e. using only the observed ratings, is presented. It is applicable as long as the weight for a particular $r_{u,i} =?$ can be expressed as a product of user and item weight:

$$w_{u,i} = \begin{cases} 1 & \text{if } r_{u,i} \neq ? \wedge r_{u,i} > 0 \\ w_u^U \cdot w_i^I & \text{if } r_{u,i} = ? \vee r_{u,i} \leq 0 \end{cases} \tag{2.63}$$

for some weight vectors $\mathbf{w}^U \in \mathbb{R}^{|U|}$ and $\mathbf{w}^I \in \mathbb{R}^{|I|}$.

This allows us to use e.g. the weighting schemes from [77] on real-world rating matrices. Furthermore, optimization procedure called the Coordinate Descent (CD) is described in [109]. CD is based on ALS, but solves the least squares problem using only one latent dimension at a time. This ensures linear scaling in the number of factors. See [109] for more about performance in terms of both computation time and quality of results.

2.4.6 Deep Learning Approaches

Together with increasing popularity of Deep Learning (DL) in the recent years, accompanied by application of deep neural networks to more and more areas of human activity [58], it is natural that many approaches have appeared aiming at application of DL methods in recommender systems. In this section, we will introduce some of the most prominent approaches.

2.4.6.1 Wide & Deep Learning For Recommender Systems

This approach has been introduced and described in [20]. The model takes available information about given $(u, i) \in U \times I$ pair and predicts the probability that i is relevant to u. While the models presented earlier in this chapter typically focus on some limited and specific type of information, such as the interaction vectors (rating-based UserKnn, ItemKnn, ARs, MF) or the attributes (demographic UserKnn, attribute-based ItemKnn), Wide & Deep models take into account a whole mixture of both the interaction and the attribute data regarding both u and i. This makes the model highly robust and capable of learning complex non-linear relationships between all mentioned.

All the input information about u and i originates from a highly sparse feature vectors $\mathbf{x}^{u,i} \in [0, 1]^d$. Individual elements of these vectors represent various data inputs:

- **User interaction data**: For each $j \in I$, there is an element $x_\ell^{u,i}$ of $\mathbf{x}^{u,i}$ storing information about $r_{u,j}$, such

$$x_\ell^{u,i} = \begin{cases} 0 & \text{if } r_{u,j} = ? \\ r_{u,j} & \text{if } r_{u,j} \neq ? \end{cases} \tag{2.64}$$

  assuming $\forall y \in 2^Y : 0 \leq \Gamma(y) \leq 1$ in Eqs. 2.2 or 2.4 in 2.1.4, whichever has been used to produce the rating matrix.

- **Item attribute data**: Tokenization process as described in 2.1.2 is used on $a^I(i)$. For each possible token $t \in \bigcup_{j \in I} \mathit{tokenization}\left(a^I(j)\right)$, there is an element $x_\ell^{u,i}$ of $\mathbf{x}^{u,i}$ encoding whether i has t in its tokenized attributes:

$$x_\ell^{u,i} = \begin{cases} 0 & \text{if } t \notin \mathit{tokenization}\left(a^I(i)\right) \\ 1 & \text{if } t \in \mathit{tokenization}\left(a^I(i)\right) \end{cases} \tag{2.65}$$

- **User attribute data**: Similarly to items, for each possible $t \in \bigcup_{v \in U} \textit{tokenization}\left(a^U(v)\right)$, there is an element $x_\ell^{u,i}$ of $\mathbf{x}^{u,i}$ encoding whether u has t:

$$x_\ell^{u,i} = \begin{cases} 0 & \text{if } t \notin \textit{tokenization}\left(a^U(u)\right) \\ 1 & \text{if } t \in \textit{tokenization}\left(a^U(u)\right) \end{cases} \tag{2.66}$$

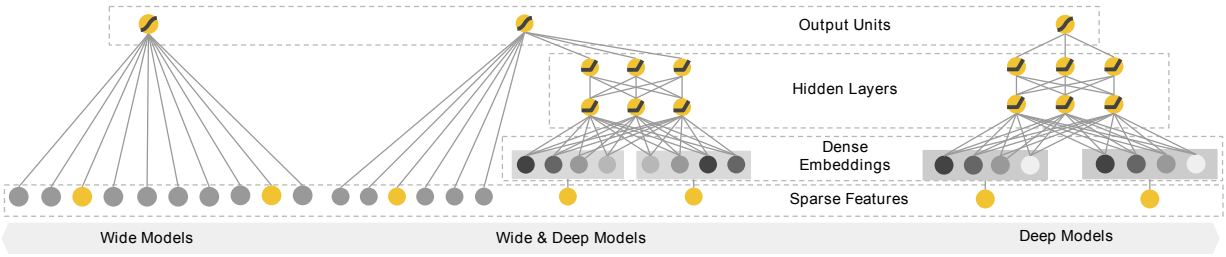

Figure 2.4: The spectrum of Wide & Deep models [20].

The structure of the model aims at balancing between memorization (learning feature co-occurrences) and generalization (learning transitions on top of feature co-occurrences). To achieve that, it combines the *wide* and the *deep* components, respectively. Both are feedforward neural networks that are trained in joint manner (Fig. 2.4).

The **wide** network can be thought as a simple generalized linear model

$$\hat{\mathbf{x}}^{u,i} \mapsto \mathbf{w}^T \cdot \hat{\mathbf{x}}^{u,i} + b \tag{2.67}$$

where $\mathbf{w}$ is the weight vector obtained while training (see below), $\hat{\mathbf{x}}^{u,i} \in [0,1]^{d+m}$ is a concatenation of $\mathbf{x}^{u,i} \in [0,1]^d$ (constructed as described above) and vector of *cross-product transformations* $(\phi_{k_1}(\mathbf{x}^{u,i}), \ldots, \phi_{k_m}(\mathbf{x}^{u,i})) \in [0,1]^m$, where

$$\phi_{k_j}(\mathbf{x}^{u,i}) = \prod_{\ell=1}^{d} \left(x_\ell^{u,i}\right)^{c_\ell^{k_j}} \tag{2.68}$$

For each transformation k_j, $\mathbf{c}^{k_j} \in \{0,1\}^d$ is a vector boolean variables representing the transformation. Effectively, $\phi_{k_j} \in [0,1]$ is a composite feature similar to product conjunction from fuzzy logic [74] of variables at positions $\left\{\ell \in \{1,\ldots,d\} \mid c_\ell^{k_j} = 1\right\}$.

While the *wide* network accepts high-dimensional, sparse vectors, the **deep** network uses low-dimensional, dense vectors on input. These vectors are obtained using autoencoder network trained on top of original sparse vectors $\mathbf{x}^{u,i}$ for various $(u,i) \in U \times I$ pairs. The autoencoder has a bottleneck of small size f (from 10 to 100), from which dense vector $\tilde{\mathbf{x}}^{u,i} \in \mathbb{R}^f$ is obtained for given $\mathbf{x}^{u,i}$. The deep network consists of n layers, where activation vector $\mathbf{a}^{(j+1)}$ of $(j+1)$-th layer computes as

$$\mathbf{a}^{(j+1)} = f^{(j)}\left(\mathbf{W}^{(j)} \cdot \mathbf{a}^{(j)} + \mathbf{b}^{(j)}\right) \tag{2.69}$$

where, denoting number of neurons in j-th layer as $N^{(j)}$, $\mathbf{a}^{(j)} \in \mathbb{R}^{N^{(j)}}$ is the activation vector from preceeding (j-th) layer $\left(\text{putting } \mathbf{a}^{(0)} = \tilde{\mathbf{x}}^{u,i}\right)$, $\mathbf{W} \in \mathbb{R}^{N^{(j+1)} \times N^{(j)}}$ is a matrix of synaptic weights, $\mathbf{b} \in \mathbb{R}^{N^{(j+1)}}$ is a vector of biases, and $f^{(j)}: \mathbb{R}^{N^{(j+1)}} \to \mathbb{R}^{N^{(j+1)}}$ is an activation function performing element-wise operations on $\left(\mathbf{W}^{(j)} \cdot \mathbf{a}^{(j)} + \mathbf{b}^{(j)}\right)$. Rectified linear units (ReLUs) [36] are proposed to be used for $f^{(j)}$:

$$\forall \ell \in \left\{1, \ldots, N^{(j+1)}\right\} : \left(f^{(j)}(\mathbf{x})\right)_\ell = \max(0, x_\ell) \tag{2.70}$$

Both the *wide* network and the *deep* network are put together using a sigmoid function, resulting in the *score* function:

$$score(u,i) = \sigma\left(\mathbf{w}_{\text{wide}} \cdot \left(x_1^{u,i}, \ldots, x_d^{u,i}, \phi_{k_1}\left(\mathbf{x}^{u,i}\right), \ldots, \phi_{k_m}\left(\mathbf{x}^{u,i}\right)\right) + \mathbf{w}_{\text{deep}}^{(n)} \cdot \mathbf{a}^{(n)} + b\right) \tag{2.71}$$

where $\sigma(x) = \frac{1}{1-e^{-x}}$ is the sigmoid function. The whole model is trained by back-propagation [87] of gradients from the output to both the wide and deep part of the model simultaneously.

2.4.6.2 AutoRec

AutoRec is a novel autoencoder framework for Collaborative Filtering presented in [95]. The model is based on training a neural network with one hidden layer, based on autoencoder paradigm (see Fig. 2.5). It exists it two symmetric variants: the User-Based AutoRec (or *U-AutoRec* for short) and the Item-Based AutoRec (or *I-AutoRec* for short). In both cases, the whole sparse vector from the rating matrix with $? \to 0$ replacements, that is either $\hat{\mathbf{r}}_{u,*}^T$ or $\hat{\mathbf{r}}_{*,i}$, is passed to the input layer, and is expected also at the output, modeled through a bottleneck of smaller size f. The authors experimented with f from 10 to 500 on different datasets.

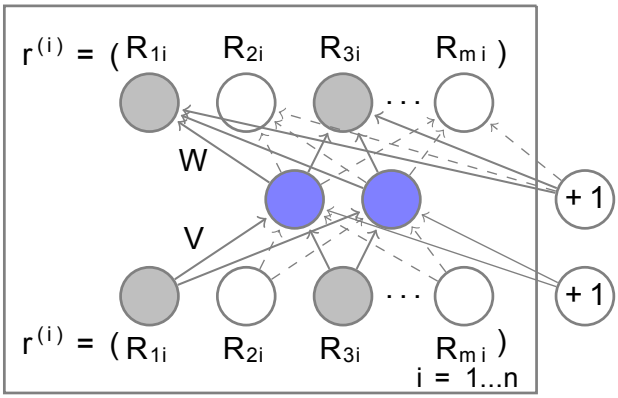

Figure 2.5: AutoRec model architecture [95].

For U-AutoRec, the *score* function is given as

$$score(u,i) = \left(h\left(\mathbf{W} \cdot g\left(\mathbf{V} \cdot \hat{\mathbf{r}}_{u,*}^T + \boldsymbol{\mu}\right) + \mathbf{b}\right)\right)_i \tag{2.72}$$

where $\hat{\mathbf{r}}_{u,*} \in \mathbb{R}^{|I|}$ is a row vector from rating matrix of user u with $? \to 0$ replacements, $\mathbf{V} \in \mathbb{R}^{f \times |I|}$ is the weights matrix for the hidden (bottleneck) layer, $\boldsymbol{\mu} \in \mathbb{R}^f$ are the biases for the hidden layer, $g\colon \mathbb{R}^f \to \mathbb{R}^f$ is the hidden layer's activation function, $\mathbf{W} \in \mathbb{R}^{|I| \times f}$ is the weights matrix for the output layer, $\mathbf{b} \in \mathbb{R}^{|I|}$ are the biases for the output layer, and $h\colon \mathbb{R}^{|I|} \to \mathbb{R}^{|I|}$ is the output layer's activation function.

Symmetrically, for I-AutoRec, the *score* is

$$score(u,i) = \left(h\left(\mathbf{W} \cdot g\left(\mathbf{V} \cdot \hat{\mathbf{r}}_{*,i} + \boldsymbol{\mu}\right) + \mathbf{b}\right)\right)_u \tag{2.73}$$

where $\hat{\mathbf{r}}_{*,i} \in \mathbb{R}^{|U|}$ is a column vector from rating matrix for item i, $\mathbf{V} \in \mathbb{R}^{f \times |U|}$ and $\mathbf{W} \in \mathbb{R}^{|U| \times f}$ are the synaptic weights matrices, $\boldsymbol{\mu} \in \mathbb{R}^f$ and $\mathbf{b} \in \mathbb{R}^{|U|}$ are the biases, and $g\colon \mathbb{R}^f \to \mathbb{R}^f$ and $h\colon \mathbb{R}^{|U|} \to \mathbb{R}^{|U|}$ are the activation functions.

In [95], only explicit rating matrices and Rating Prediction task are considered, which affects how the network is trained. Given rating matrix $\mathbf{R}^{|U| \times |I|}$, for U-AutoRec the loss function of the optimization problem is given as

$$\min_{\substack{\mathbf{V} \in \mathbb{R}^{f \times |I|} \\ \mathbf{W} \in \mathbb{R}^{|I| \times f} \\ \boldsymbol{\mu} \in \mathbb{R}^f \\ \mathbf{b} \in \mathbb{R}^{|I|}}} \sum_{\substack{u \in U \\ i \in I \\ r_{u,i} \neq ?}} \left(r_{u,i} - \left(h\left(\mathbf{W} \cdot g\left(\mathbf{V} \cdot \hat{\mathbf{r}}_{u,*}^T + \boldsymbol{\mu}\right) + \mathbf{b}\right)\right)_i\right)^2 + \frac{\lambda}{2} \cdot \left(\|\mathbf{V}\|^2 + \|\mathbf{W}\|^2\right) \tag{2.74}$$

and for I-AutoRec as

$$\min_{\substack{\mathbf{V} \in \mathbb{R}^{f \times |U|} \\ \mathbf{W} \in \mathbb{R}^{|U| \times f} \\ \boldsymbol{\mu} \in \mathbb{R}^f \\ \mathbf{b} \in \mathbb{R}^{|U|}}} \sum_{\substack{u \in U \\ i \in I \\ r_{u,i} \neq ?}} \left(r_{u,i} - \left(h\left(\mathbf{W} \cdot g\left(\mathbf{V} \cdot \hat{\mathbf{r}}_{*,i} + \boldsymbol{\mu}\right) + \mathbf{b}\right)\right)_u\right)^2 + \frac{\lambda}{2} \cdot \left(\|\mathbf{V}\|^2 + \|\mathbf{W}\|^2\right) \tag{2.75}$$

The parameters $\mathbf{V}$, $\mathbf{W}$, $\boldsymbol{\mu}$, and $\mathbf{b}$ are trained using backpropagation [87]. Similarly to MF, there is a regularization parameter λ to prevent overfitting. Omitting $r_{u,i} \neq ?$ in the loss function means that only the weights associated with observed ratings are updated.

Considering implicit feedback matrices, it is natural idea to extend ideas from [47] (imputation of zeros with confidence-based loss weighting as in Eq. 2.56) and [100] (imputation of optimized values with (u,i)-dependent error weight). This has been proposed in [60].

Mentioning MF, it is especially noteworthy that using identity functions $g(\mathbf{x}) = \mathbf{x}$ and $h(\mathbf{y}) = \mathbf{y}$ makes AutoRec roughly equivalent to MF with biases as proposed in [56] and shown in Eq. 2.50.

With identity activation functions, considering U-AutoRec, Eq. 2.72 reduces to

$$score(u,i) = \left(\mathbf{W} \cdot \left(\mathbf{V} \cdot \hat{\mathbf{r}}_{u,*}^T + \boldsymbol{\mu}\right) + \mathbf{b}\right)_i \tag{2.76}$$

Substituting $\mathbf{W} \in \mathbb{R}^{|I| \times f}$ with $\mathbf{Q}^T \in \mathbb{R}^{|I| \times f}$ from Eq. 2.50 and $\left(\mathbf{V} \cdot \hat{\mathbf{R}}^T\right) \in \mathbb{R}^{f \times |U|}$ with $\mathbf{P} \in \mathbb{R}^{f \times |U|}$ from 2.50, we get

$$score(u,i) = \left(\mathbf{Q}^T \cdot (\mathbf{p}_{*,u} + \boldsymbol{\mu}) + \mathbf{b}\right)_i = \mathbf{q}_{*,i}^T \cdot \mathbf{p}_{*,u} + \mathbf{q}_{*,i}^T \cdot \boldsymbol{\mu} + b_i \tag{2.77}$$

which is form very similar to 2.50. The same holds for I-AutoRec with reduced form $score(u,i) = (\mathbf{W} \cdot (\mathbf{V} \cdot \hat{\mathbf{r}}_{*,i} + \boldsymbol{\mu}) + \mathbf{b})_u$, where substituting $\mathbf{W} \in \mathbb{R}^{|U| \times f}$ with $\mathbf{P}^T \in \mathbb{R}^{|U| \times f}$ and $\left(\mathbf{V} \cdot \hat{\mathbf{R}}\right) \in \mathbb{R}^{f \times |I|}$ with $\mathbf{Q} \in \mathbb{R}^{f \times |I|}$ yields $\left(\mathbf{P}^T \cdot (\mathbf{q}_{*,i} + \boldsymbol{\mu}) + \mathbf{b}\right)_u = \mathbf{p}_{*,u}^T \cdot \mathbf{q}_{*,i} + \mathbf{p}_{*,u}^T \cdot \boldsymbol{\mu} + b_u$.

Considering the above, AutoRec brings generalization on top of MF only when non-linear activation functions are provided. Several experiments are done in [95], showing particularly the importance of non-linearity of g for the hidden layer. Based on the experimentally obtained results, sigmoid function $\left(\frac{1}{1-e^{-x}}\right)$ for g and identity function for h were stated as optimal. In the experiments performed in this thesis, we are using hyperbolic tangent (tanh) for g, because several advantages has been reported for tanh over sigmoid thanks to its centering around 0 [35].

2.5 Evaluation in Recommender Systems

While the research in the recent years has been highly focused on recommendation algorithms/models. All the models share a common goal: help users discover items that the will like. This may include they will find them interesting, enjoy them, or will be interested in consuming them, where "consuming" may stay for reading (online news, blog posts, books), playing (audio/video streaming, gaming), installing (applications and plugins), purchasing (E-Commerce), attending (courses, E-learning), visiting (cultural events and venues), etc. This closely follows the business goals of RSs, which may vary from simply improving user experience and satisfaction, decrease time spent searching for a relevant content or shortening purchase cycle, increasing the amount/total price of items that the users purchase, increasing number of pages they visit (often hand in hand with more advertisements being displayed), etc.

These high-level goals are common in practice, but it is not easy to set a clear performance measure for them that would allow comparing them to each other. Difficulties especially arise in the field of research, where a well-defined and measurable performance criteria are crucial to show how well the proposed method works. Moreover, it is quite common that researchers don't have access to production systems, where the users would be in interaction with the models, allowing to collect immediate feedback from them. This led to a whole line of approximative, yet mathematically well-defined measures, with intuition behind them that maximizing/minimizing them *should* lead to better satisfaction of the aforementioned goals.

From a high-level perspective, we may distinguish between two main classes of such measures: the *offline measures* and the *online measures*.

The **offline measures** as crucial for research, as well for searching for good models and their parameterizations, without the need to deploy them to production and risk exposing the users to recommendations of poor quality in case of model malfunction. All that is needed for such evaluation is the dataset of already collected historical interactions. Techniques like split or cross validation have predominant position here. It can be said

that these measures are relatively well-studied, and that there is some elementary consensus among researchers on a set of measures that make the most sense.

On the contrary, the **online measures** require production deployment. Real users have to interact in live environment with the deployed models, so we can estimate model quality based on user feedback or behavior. However, not all researchers have access to such environments. Furthermore, online experiments are unreproducible from their very nature, undermining the need for repeatability of scientific results. They are executed during specific timeframe and circumstances that will not be repeated. For example, running an online experiment on a news portal will yield results that are specific for the actual social/political/cultural events that happened during the experiment and resulted in specific set and characteristics of the published articles (items) available, not even mentioning possible shifts in structure and social mood of the userbase over time.

The measures themselves reflect the recommendation task (see 2.2) that the system or model solves. We will discuss individual task-specific measures below in this chapter.

2.5.1 Split and Cross Validation

Provided a recommendation dataset, standard way of measuring the performance of the training algorithm (that is algorithm which constructs the model, as defined in 2.2) is to use the split or the cross validation.

In **split validation**, considering training set $\mathcal{T} \subseteq \mathcal{X} \times \mathcal{Y}$ (as introduced in Sec. 2.2.1), only a random subset $\mathcal{T}_{\text{train}} \subset \mathcal{T}$ is selected and passed to learning algorithm, while the rest is left apart. Typically, $\mathcal{T} \setminus \mathcal{T}_{\text{train}}$ is further divided into the **validation set** $\mathcal{T}_{\text{val}}$ and **test set** $\mathcal{T}_{\text{test}}$ such that $\mathcal{T}_{\text{val}} \cap \mathcal{T}_{\text{test}} = \emptyset$ and $\mathcal{T}_{\text{val}} \cup \mathcal{T}_{\text{test}} = \mathcal{T} \setminus \mathcal{T}_{\text{train}}$.

Validation set is used to optimize hyperparameterization $P \in \mathcal{P}_{\mathcal{A}}$ of $\mathcal{A}$. Repeatedly, for different hyperparameterizations $P \in \mathcal{P}_{\mathcal{A}}$, in compliance with Eq. 2.7, $\mathcal{A}(P, \mathcal{T}_{\text{train}})$ is executed to produce different candidate model $m_P \colon \mathcal{X} \to \mathcal{Y}$. Individual models are then evaluated on $\mathcal{T}_{\text{val}}$ in such a way that for each $m_P \in \{\mathcal{A}(P, \mathcal{T}_{\text{train}}) \mid P \in \mathcal{P}_{\mathcal{A}}\}$, all the validation examples $(\mathbf{x}_\ell, \mathbf{y}_\ell) \in \mathcal{T}_{\text{val}}$ are taken and for each example, $m_P(\mathbf{x}_\ell)$ is compared to known $\mathbf{y}_\ell$.

The comparison is typically done using a **loss function** $L \colon \mathcal{Y}^2 \to \mathbb{R}$ which expresses the difference (error) between $\mathbf{y}_\ell$ and $m(\mathbf{x}_\ell)$ as a real number. The lower value of $L(\hat{y}, y)$, the lower error and hence the better. Considering the validation set $\mathcal{T}_{\text{val}}$, it is useful to define a **validation loss** function on the whole $\mathcal{T}_{\text{val}}$ as

$$\mathcal{L} \colon \mathcal{Y}^{\mathcal{X}} \times 2^{\mathcal{X} \times \mathcal{Y}} \to \mathbb{R} \tag{2.78}$$

which takes a model m and set of validation examples $\mathcal{T}_{\text{val}}$ and produces an aggregated loss as a real number. One possible implementations of $\mathcal{L}$ may be simple sum

$$\mathcal{L}(m, \mathcal{T}_{\text{val}}) = \sum_{(\mathbf{x}_\ell, \mathbf{y}_\ell) \in \mathcal{T}_{\text{val}}} L(m(\mathbf{x}_\ell), \mathbf{y}_\ell) \tag{2.79}$$

but more sophisticated functions are often used as will be presented later in this section.

For a given learning algorithm $\mathcal{A}$, optimal hyperparameterization $P^*_{\mathcal{A}} \in \mathcal{P}_{\mathcal{A}}$ is then determined as

$$P^*_{\mathcal{A}} = \arg\min_{P \in \mathcal{P}_{\mathcal{A}}} \mathcal{L}\left(\mathcal{A}\left(P, \mathcal{T}_{\text{train}}\right), \mathcal{T}_{\text{val}}\right) \tag{2.80}$$

Complementarily to using loss function which evaluates the model in negative nature by error, we can also use **accuracy function** $F\colon \mathcal{Y}^2 \to \mathbb{R}$ which expresses the similarity between $\mathbf{y}_\ell$ and $m(\mathbf{x}_\ell)$ in terms of positive utility. Considering whole $\mathcal{T}_{\text{val}}$, a **validation reward** function can be defined as

$$\mathcal{F}\colon \mathcal{Y}^{\mathcal{X}} \times 2^{\mathcal{X} \times \mathcal{Y}} \to \mathbb{R} \tag{2.81}$$

with optimal hyperparameterization $P^*_{\mathcal{A}} \in \mathcal{P}_{\mathcal{A}}$ given as

$$P^*_{\mathcal{A}} = \arg\max_{P \in \mathcal{P}_{\mathcal{A}}} \mathcal{F}\left(\mathcal{A}\left(P, \mathcal{T}_{\text{train}}\right), \mathcal{T}_{\text{val}}\right) \tag{2.82}$$

Validation set is used for tuning the hyperparameterization of one particular learning algorithm $\mathcal{A}$. When a whole set of algorithms $\mathfrak{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_r\}$ is available, not yet used test set $\mathcal{T}_{\text{test}}$ is utilized to do the cross algorithm comparison. Given validation loss function $\mathcal{L}$ for a model $m \in \mathcal{Y}^{\mathcal{X}}$ as defined in Eq. 2.78, we can also define *validation loss of algorithm* $\mathfrak{L}(\mathcal{A})$ for given $\mathcal{A} \in \mathfrak{A}$ as the overall loss on $\mathcal{T}_{\text{test}}$ of a model trained on the whole $\mathcal{T}_{\text{train}} \cup \mathcal{T}_{\text{val}}$ using optimally hyperparameterized $\mathcal{A}$:

$$\mathfrak{L}(\mathcal{A}) = \mathcal{L}\left(\mathcal{A}\left(P^*_{\mathcal{A}}, \mathcal{T}_{\text{train}} \cup \mathcal{T}_{\text{val}}\right), \mathcal{T}_{\text{test}}\right) \tag{2.83}$$

Symmetrically, *validation reward of algorithm* $\mathfrak{F}\left(\mathcal{A}\right)$ for given $A \in \mathfrak{A}$ can be defined as:

$$\mathfrak{F}(\mathcal{A}) = \mathcal{F}\left(\mathcal{A}\left(P^*_{\mathcal{A}}, \mathcal{T}_{\text{train}} \cup \mathcal{T}_{\text{val}}\right), \mathcal{T}_{\text{test}}\right) \tag{2.84}$$

Finally, for given dataset $\mathcal{T}$, algorithm loss function $\mathfrak{L}$ (or reward function $\mathfrak{F}$), and set of available algorithms $\mathfrak{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_r\}$, optimal algorithm $\mathcal{A}^*$ is defined as $\mathcal{A}^* = \arg\min_{\mathcal{A} \in \mathfrak{A}} \mathfrak{L}\left(\mathcal{A}\right)$ or, symmetrically, as $\mathcal{A}^* = \arg\max_{\mathcal{A} \in \mathfrak{A}} \mathfrak{F}\left(\mathcal{A}\right)$. It is the search for couple $(\mathcal{A}^*, P^*_{\mathcal{A}^*})$ which defines optimization problem that data scientists are typically solving for given $\mathcal{T}$ and $\mathcal{L}$ (or $\mathcal{F}$).

The mechanism of splitting $\mathcal{T}$ into $\mathcal{T}_{\text{train}}$, $\mathcal{T}_{\text{val}}$, and $\mathcal{T}_{\text{test}}$, typically consists of putting all $(\mathbf{x}_\ell, \mathbf{y}_\ell) \in \mathcal{T}$ into a list, random shuffling that list, and splitting the shuffled list using ratios such as 80:10:10. The actual numbers depend mostly of the size of the dataset with the aim of getting stable enough estimate. For very large datasets consisting of billions of training examples, ratios like 98:1:1 may be sufficient, considering especially the combinatoric explosion in size of $\mathcal{P}_{\mathcal{A}}$ if it consists of multiple independent hyperparameters (such as f and λ in MF), and the computation time needed to evaluate the trained model on each example.

The motivation for introducing $\mathcal{T}_{\text{test}}$ besides $\mathcal{T}_{\text{val}}$ (instead of partitioning only into $\mathcal{T}_{\text{train}}$ and $\mathcal{T}_{\text{val}}$), comes from the fact that the split is done randomly, and there is a risk of hyper-overfitting the algorithm (through hyperparameterization search) towards particular split.

This may lead to performance overestimation for determined $(\mathcal{A}, P_{\mathcal{A}^*})$. Evaluation on $\mathcal{T}_{\text{test}}$ is meant as unbiased performance estimate of the resulting $\mathcal{A}(P^*_{\mathcal{A}}, \mathcal{T})$ model.

To overcome the general hyper-overfitting risk connected with split validation, k-fold **cross validation** is often the more preferred approach. $\mathcal{T}$ is randomly partitioned into k equally-sized sets $\mathfrak{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ such that $\forall i, j \in \{1, \ldots, k\}: \mathcal{T}_i \cap \mathcal{T}_j = \emptyset \wedge ||\mathcal{T}_i| - |\mathcal{T}_j|| \leq 1$ and $\bigcup_{i=1}^{k} \mathcal{T}_i = \mathcal{T}$. The performance is estimated as an average over k rounds, taking $\mathcal{T}_j$ as the validation set and $\bigcup_{\mathcal{T}_j \in \mathfrak{T} \setminus \{\mathcal{T}_i\}} \mathcal{T}_j$ as the training set at the i-th round. Given validation loss function $\mathcal{L}: \mathcal{Y}^{\mathcal{X}} \times 2^{\mathcal{X} \times \mathcal{Y}} \to \mathbb{R}$ (Eq. 2.78), learning algorithm $\mathcal{A}$ hyperparameterized by $P \in \mathcal{P}_{\mathcal{A}}$, and a training set $\mathcal{T}$ with cross-validation partition $\mathfrak{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$, we may define **cross-validation loss** $\mathcal{L}_{\text{XV}}(\mathcal{A}, P)$ as

$$\mathcal{L}_{\text{XV}}(\mathcal{A}, P) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\left(\mathcal{A}\left(P, \cup_{\mathcal{T}_j \in \mathfrak{T} \setminus \{\mathcal{T}_i\}} \mathcal{T}_j\right), \mathcal{T}_i\right) \tag{2.85}$$

and, symmetrically for validation reward function $\mathcal{F}: \mathcal{Y}^{\mathcal{X}} \times 2^{\mathcal{X} \times \mathcal{Y}} \to \mathbb{R}$ (Eq. 2.81), cross-validation reward $\mathcal{F}_{\text{XV}}(\mathcal{A}, P)$ as

$$\mathcal{F}_{\text{XV}}(\mathcal{A}, P) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{F}\left(\mathcal{A}\left(P, \cup_{\mathcal{T}_j \in \mathfrak{T} \setminus \{\mathcal{T}_i\}} \mathcal{T}_j\right), \mathcal{T}_i\right) \tag{2.86}$$

Using cross-validation, additional test set it typically not necessary as the method is robust enough to provent hyper-overfitting.

2.5.2 Rating Prediction Measures

As discussed in previous sections of this chapter, rating prediction is popular task in research related to Collaborative Filtering. Being slightly biased towards publicly available datasets missing implicit feedback, it strictly assumes explicit rating matrices. Since $m^{\text{RP}}(u, i)$, as defined in Sec. 2.2.2, returns real number as an estimate of $r_{u,i}$ in the rating matrix, rating prediction measures have typically form of validation loss functions (as in Eq. 2.78) measuring the numerical predictive accuracy.

Following terminology from Secs. 2.2.1 and 2.5.1, given rating matrix $\mathbb{R} \in (\mathbb{R} \cup \{?\})^{U \times I}$ trivially induces a training set

$$\mathcal{T} = \{((u, i), r_{u,i}) \mid u \in U \wedge i \in I \wedge r_{u,i} \neq ?\} \tag{2.87}$$

The most popular methodology lies in randomly splitting this training set into $\mathcal{T}_{\text{train}}$ and $\mathcal{T}_{\text{val}}$ as in 2.5.1. It is noteworthy that it is completely random subset of the whole explicit rating matrix for different observed (u, i) pairs. For rating prediction models (Eq. 2.8) $m_{\text{RP}}: U \times I \to \mathbb{R}$, the measures are given as validation loss functions $\mathcal{L}_{\text{RP}}: \mathbb{R}^{(U \times I)} \times 2^{U \times I \times \mathbb{R}} \to \mathbb{R}$.

2.5.2.1 Mean Absolute Error

One of the simplest validation loss functions is the Mean Absolute Error (MAE), which simply averages individual deviations of predicted ratings from the true ratings in the validation set. The validation loss function is defined as

$$\mathrm{MAE}\left(m_{\mathrm{RP}}, \mathcal{T}_{\mathrm{val}}\right) = \frac{1}{|\mathcal{T}_{\mathrm{val}}|} \sum_{((u,i),r_{u,i})\in\mathcal{T}_{\mathrm{val}}} |m_{\mathrm{RP}}(u,i) - r_{u,i}| \tag{2.88}$$

MAE was popular in earlier stages of research in Collaborative Filtering [93, 94, 81, 70, 1], all before 2006, when the Netflix Prize Challenge took place and the focus has been shifted towards RMSE (see below).

In [38], Normalized Mean Absolute Error (NMAE) has been proposed to allow comparison between different datasets (the authors used rating scale from -10 to 10, while datasets like MovieLens used ratings from 1 to 5). Given training dataset $\mathcal{T}$, NMAE validation loss function is given simply as

$$\mathrm{NMAE}\left(m_{\mathrm{RP}}, \mathcal{T}_{\mathrm{val}}\right) = \frac{\mathrm{MAE}\left(m_{\mathrm{RP}}, \mathcal{T}_{\mathrm{val}}\right)}{\max\limits_{((u,i),r_{u,i})\in\mathcal{T}}(r_{u,i}) - \min\limits_{((u,i),r_{u,i})\in\mathcal{T}}(r_{u,i})} \tag{2.89}$$

2.5.2.2 (Root) Mean Squared Error

Alternatively to MAE, which gives all the errors equal weights, Mean Squared Error (MSE) can be used. It increases the impact of higher errors, and provides provides more mathematical elegance and compatibility with numerical optimization methods involved in e.g. MF optimization tasks (Eqs. 2.47, 2.51, 2.53). MSE validation loss function is given as

$$\mathrm{MSE}\left(m_{\mathrm{RP}}, \mathcal{T}_{\mathrm{val}}\right) = \frac{1}{|\mathcal{T}_{\mathrm{val}}|} \sum_{((u,i),r_{u,i})\in\mathcal{T}_{\mathrm{val}}} \left(m_{\mathrm{RP}}(u,i) - r_{u,i}\right)^2 \tag{2.90}$$

Despite used in some studies (e.g. [21, 42]), MSE has not received much popularity. On the contrary, Root Mean Squared Error (RMSE), given as

$$\mathrm{RMSE}\left(m_{\mathrm{RP}}, \mathcal{T}_{\mathrm{val}}\right) = \sqrt{\mathrm{MAE}\left(m_{\mathrm{RP}}, \mathcal{T}_{\mathrm{val}}\right)} \tag{2.91}$$

became absolutely dominant evaluation method for Rating Prediction tasks after the Netflix Prize Challenge organized in 2006, where it was used as the only evaluation criterion. Since then RMSE have been adopted by the research community and used for comparison between newly introduced algorithms [11, 55, 95, 80, 68]. Similarly to MSE, RMSE puts emphasis on higher errors, but allows more intuitive interpretation of the resulting value, being it on the same scale of the original ratings.

2.5.3 Binary Classification Measures

While most of the earlier research in Recommender Systems mainly focused on the Rating Prediction task, many publications started to appear over time pointing out the fact that recommendation can also be thought as binary classification. Already in 1998, transformation of explicit rating matrix into a binary one, using either technique similar to Eq. 2.9 [13], or by setting user-dependent threshold [8], has been proposed, opening the possibilities of using measures from Information Retrieval, mainly the *precision*, *recall*, and the *F-Measure* (see below). Many researchers followed this: precision and recall e.g. in the original paper presenting matrix factorization [92].

Considering binary rating matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{U \times I}$, we may consider training sets

$$\mathcal{T} = \{((u,i), r_{u,i}) \mid u \in U \wedge i \in I \wedge r_{u,i} \neq ?\} \tag{2.92}$$

or

$$\mathcal{T} = \{((u,i), 0) \mid u \in U \wedge i \in I \wedge r_{u,i} \in \{0, ?\}\} \cup \{((u,i), 1) \mid u \in U \wedge i \in I \wedge r_{u,i} = 1\} \tag{2.93}$$

depending on whether unobserved ratings are treated as zeros (unless rated, items are considered irrelevant by default) or not.

2.5.3.1 Precision, Recall, F-Measure

In Binary Classification, the measures are mostly given as validation reward functions. In compliance with Eq. 2.81, we may define **precision** as portion of truly relevant items in all the items that were predicted relevant by the model:

$$\mathit{precision}\,(m, \mathcal{T}_{\text{val}}) = \frac{|\{((u,i), r_{u,i}) \in \mathcal{T}_{\text{val}} \mid m(u,i) = 1 \wedge r_{u,i} = 1\}|}{|\{((u,i), r_{u,i}) \in T_{\text{val}} \mid m(u,i) = 1\}|} \tag{2.94}$$

Similarly, we may define the **recall** as the portion of all relevant items that were really predicted relevant:

$$\mathit{recall}\,(m, \mathcal{T}_{\text{val}}) = \frac{|\{((u,i), r_{u,i}) \in \mathcal{T}_{\text{val}} \mid m(u,i) = 1 \wedge r_{u,i} = 1\}|}{|\{((u,i), r_{u,i}) \in T_{\text{val}} \mid r_{u,i} = 1\}|} \tag{2.95}$$

Precision is sometimes referred to as the *positive predictive value (PPV)* and recall is sometimes referred to as the *true positive rate (TPR)* or the *sensitivity*.

Precision and recall are complementary to each other and each brings specific information. Constant model $m(u,i) = 1$ will have $\mathit{recall}\,(m, \mathcal{T}_{\text{val}}) = 1$, but very low precision. To address this fact, complementary measures have been invented. Of the most prominent, the **F-Measure** can be defined as validation reward function combining both the precision and the recall in form of harmonic mean:

$$F_1\,(m, \mathcal{T}_{\text{val}}) = \frac{2 \cdot \mathit{precision}\,(m, \mathcal{T}_{\text{val}}) \cdot \mathit{recall}\,(m, \mathcal{T}_{\text{val}})}{\mathit{precision}\,(m, \mathcal{T}_{\text{val}}) + \mathit{recall}\,(m, \mathcal{T}_{\text{val}})} \tag{2.96}$$

2.5.3.2 Area Under Receiver Operating Characteristic Curve

As suggested in [44], **Area Under Receiver Operating Characteristic Curve (AUC ROC)** provides a theoretically grounded alternative to precision and recall. In Sec. 2.4, we introduced several recommendation models under the common framework of the $score\colon U \times I \to \mathbb{R}$ function. In Eq. 2.16, we showed that using $score$, we can easily build binary classification model using threshold $\theta \in \mathbb{R}$ such that the model predicts as relevant for user $u \in U$ exactly those items $i \in I$ for which $score(u,i) \geq \theta$. Thinking about θ as a continuous parameter, uncountably infinite set of binary classification models can be built on top of $score$. Assuming $\forall i \in I, u \in U\colon \theta_{\min} \leq score(u,i) \leq \theta_{\max}$, AUC ROC is a measure for evaluating this whole set of models for $\theta \in [\theta_{\min}, \theta_{\max}]$.

Besides already defined validation reward function $recall$, let us further define validation loss (as in Eq. 2.78) function $fallout\colon \{0,1\}^{(U\times I)} \times 2^{U\times I\times\{0,1\}} \to \mathbb{R}$ as the portion of irrelevant items that were falsely predicted as relevant:

$$fallout = (m, \mathcal{T}_{\text{val}}) = \frac{\left|\left\{\left((u,i), r_{u,i}\right) \in T_{\text{val}} \mid m(u,i) = 1 \wedge r_{u,i} = 0\right\}\right|}{\left|\left\{\left((u,i), r_{u,i}\right) \in T_{\text{val}} \mid r_{u,i} = 0\right\}\right|} \tag{2.97}$$

Given a training set $\mathcal{T}$ as in Eq. 2.92 or in Eq. 2.93 partitioned into $\mathcal{T}_{\text{train}}$ and $\mathcal{T}_{\text{val}}$ as in Sec. 2.5.1, and a function $score\colon U \times I \to \mathbb{R}$ (assuming the function has been somehow constructed using $\mathcal{T}_{\text{train}}$), let us introduce a binary classification learning algorithm (as in Eq. 2.10) $\mathcal{A}^{\text{BC}}_{score}$ with hyperparameterization $\mathcal{P}_{\mathcal{A}^{\text{BC}}_{score}} = [\theta_{\min}, \theta_{\max}] \subseteq \mathbb{R}$ as:

$$\mathcal{A}^{\text{BC}}_{score}\left(\theta, \mathcal{T}_{\text{train}}\right) = \left((u,i) \mapsto \begin{cases} 0 & score(u,i) < \theta \\ 1 & score(u,i) \geq \theta \end{cases}\right) \tag{2.98}$$

Because $\mathcal{T}_{\text{train}}$ is not used while building model (we assume that the knowledge from it is somehow captured in provided function $score$), let us simply write $\mathcal{A}^{\text{BC}}_{score}(\theta)$ instead of the full $\mathcal{A}^{\text{BC}}_{score}(\theta, \mathcal{T}_{\text{train}})$. It should be obvious that for given $\theta \in [\theta_{\min}, \theta_{\max}]$, we obtain a pair $\left(fallout\left(\mathcal{A}^{\text{BC}}_{score}(\theta), \mathcal{T}_{\text{val}}\right), recall\left(\mathcal{A}^{\text{BC}}_{score}(\theta), \mathcal{T}_{\text{val}}\right)\right) \in [0,1]^2$ with the most desirable point at $[0,1]$ (only true positives without false negatives), and that both $fallout\left(\mathcal{A}^{\text{BC}}_{score}(\theta), \mathcal{T}_{\text{val}}\right)$ and $recall\left(\mathcal{A}^{\text{BC}}_{score}(\theta), \mathcal{T}_{\text{val}}\right)$ are monotonically increasing with θ. In case of fallout, it holds

$$\forall \theta, \theta' \in [\theta_{\min}, \theta_{\max}]\colon \theta > \theta' \implies fallout\left(\mathcal{A}^{\text{BC}}_{score}(\theta), \mathcal{T}_{\text{val}}\right) \geq fallout\left(\mathcal{A}^{\text{BC}}_{score}(\theta'), \mathcal{T}_{\text{val}}\right) \tag{2.99}$$

because setting higher threshold, we can never produce less positives ((u,i) pairs predicted relevant), and hence we can neither get less false positives ((u,i) pairs where $r_{u,i} \neq 1$ but the prediction is falsely relevant). Similarly, in case of recall, it holds

$$\forall \theta, \theta' \in [\theta_{\min}, \theta_{\max}]\colon \theta > \theta' \implies recall\left(\mathcal{A}^{\text{BC}}_{score}(\theta), \mathcal{T}_{\text{val}}\right) \geq recall\left(\mathcal{A}^{\text{BC}}_{score}(\theta'), \mathcal{T}_{\text{val}}\right) \tag{2.100}$$

because again, greater or equally sized set of positives cannot lead to smaller set of true positives ((u,i) pairs where $r_{u,i} = 1$ and the prediction really is 1). As a consequence of Eqs. 2.99 and 2.100, recall is monotonically increasing with fallout, that is

$$\begin{aligned} \forall \theta, \theta' \in [\theta_{\min}, \theta_{\max}]\colon & fallout\left(\mathcal{A}^{\text{BC}}_{score}(\theta), \mathcal{T}_{\text{val}}\right) > fallout\left(\mathcal{A}^{\text{BC}}_{score}(\theta'), \mathcal{T}_{\text{val}}\right) \\ & \implies recall\left(\mathcal{A}^{\text{BC}}_{score}(\theta), \mathcal{T}_{\text{val}}\right) \geq recall\left(\mathcal{A}^{\text{BC}}_{score}(\theta'), \mathcal{T}_{\text{val}}\right) \end{aligned} \tag{2.101}$$

Set of points $X = \left\{ \left(\mathit{fallout}\left(\mathcal{A}^{\mathrm{BC}}_{\mathit{score}}(\theta), \mathcal{T}_{\mathrm{val}}\right), \mathit{recall}\left(\mathcal{A}^{\mathrm{BC}}_{\mathit{score}}(\theta), \mathcal{T}_{\mathrm{val}}\right)\right) \mid \theta \in [\theta_{\min}, \theta_{\max}] \right\}$ hence forms a (discontinuous) curve in the fallout-recall $[0,1]^2$ plane, starting at $[0,0]$ for $\theta = \theta_{\min}$ and ending at $[1,1]$ for $\theta = \theta_{\max}$. Area under this curve gives us the quality of the provided *score* function as:

$$AUC_{\mathrm{ROC}}(\mathit{score}) = \int_0^1 \left(\underset{\substack{(\mathit{fallout},\mathit{recall})\in X \\ \text{w.r.t. } \mathit{fallout}\leq f}}{\arg\max} \mathit{fallout} \right)_2 df \tag{2.102}$$

The formula reflects the fact that X is not continuous ($U \times I$ is finite), and hence when integrating by fallout f, the recall value associated with arbitrary $f \in [0,1]$ is given as the *recall* of the point in $(\mathit{fallout}, \mathit{recall}) \in X$ with the greatest $\mathit{fallout} \leq y$. It holds $0 \leq AUC_{\mathrm{ROC}}(\mathit{score}) \leq 1$ with 0 being worst possible and 1 the best possible value.

2.5.4 Top-N Recommendation Measures

As we stated earlier, Top-N Recommendation (Sec. 2.2.4) is one the most important and practical recommendation tasks. User is presented N items and it is desirable for as many of them as possible to be relevant. A lot of measures were hence been researched and suggested for recommendation sets of fixed size N (typically ranging from 5 to 30). There exist measures expressing accuracy and relevancy of the recommend items to given user, but other measures, focusing mainly on the overall composition of the recommended items across the whole userbase, exist as well. We will summarize them in this section.

2.5.4.1 Precision@N, Recall@N

While the precision and the recall as defined in Eqs. 2.94 and 2.95 comply with general Binary Classification task, they are far from being practical in their pure sense [44]. Many authors noticed that, proposing binary classification measures which take set of items of size N into account [13, 92, 23, 100]. Measuring precision and recall on such sets is known from Information Retrieval as *precision at k* (most often written at precision@k or precision@10 for specific $k = 10$) and *recall at k* (written as recall@k). To avoid confusion between the letters k (Information Retrieval) and N (Top-N in Recommender Systems), we will use the terms precision@N and recall@N.

Considering construction of the training set $\mathcal{T}$ as in 2.5.1, the situation is not as straightfoward as in Rating Prediction or simple Binary Classification. Assuming there is binary rating matrix $\mathbf{R} \in \{0, 1, ?\}^{U \times I}$ provided, one of the possibilities is processing the matrix in user-wise manner, and create *observation* ($\mathbf{x}$) and *target* ($\mathbf{y}$) set for each user $u \in U$ by randomly partitioning the set of items which are relevant for that user. In other words, denoting $r^+(u) = \{i \in I \mid r_{u,i} = 1\}$ as the set of relevant items for user $u \in U$, the training set can be defined as follows:

$$\mathcal{T} = \left\{ ((u, \mathit{obs}), \mathit{target}) \mid u \in U \wedge \mathit{obs} \cup \mathit{target} = r^+(u) \wedge \mathit{obs} \cap \mathit{target} = \emptyset \wedge \mathit{obs} \neq \emptyset \wedge |\mathit{target}| = N \right\} \tag{2.103}$$

Note the $|target|= N$ condition: We want non-empty observation set and target set of size at least N. Considering $N = 5$ or $N = 10$, this may be practical on academic datasets, but on implicit rating datasets with anonymous sessions in industrial practice, it is very common that $|r^+(u)|$ often follows power-law distribution, with vast majority of users having ≤ 3 relevant items in their history. We may then relax on such condition, which may result in increase in size of $\mathcal{T}$ in order of magnitude on industrial datasets:

$$\mathcal{T} = \left\{((u, obs), target) \mid u \in U \wedge obs \cup target = r^+(u) \wedge obs \cap target = \emptyset \wedge obs \neq \emptyset\right\} \quad (2.104)$$

Given $\mathcal{T}_{\text{val}} \subseteq (U \times 2^I) \times 2^I$ as in Eqs. 2.103 or 2.104, and a Top-N recommendation model m modified against Eq. 2.11 in the sense that for user u, it only takes observation set obs of relevant items into account ($m{:}\, U \times 2^I \to \{I' \subset I \mid |I'| = N\}$), we can define **precision@**N as a validation reward function (in compliance with Eq. 2.81) as

$$precision@N\,(m, \mathcal{T}_{\text{val}}) = \frac{1}{|\mathcal{T}_{\text{val}}|} \sum_{((u,obs),target)\in\mathcal{T}_{\text{val}}} \frac{|m(u, obs) \cap target|}{|m(u, obs)|} \quad (2.105)$$

For each user, we let the model recommend N items and compute the portion of them that really were relevant (that is, they were in set of hidden items *target*). The result is then average over all the users. Similarly, we can define **recall@**N as a validation reward function which computes the portion of all the hidden revelant items in *target* that were recommended by the model as

$$recall@N\,(m, \mathcal{T}_{\text{val}}) = \frac{1}{|\mathcal{T}_{\text{val}}|} \sum_{((u,obs),target)\in\mathcal{T}_{\text{val}}} \frac{|m(u, obs) \cap target|}{|target|} \quad (2.106)$$

In case of training set as in Eq. 2.103, where $\forall((u, obs), target) \in \mathcal{T}_{\text{val}}{:}\, |target|= N$, it holds $precision@N\,(m, \mathcal{T}_{\text{val}}) = recall@N\,(m, \mathcal{T}_{\text{val}})$, because recommending $N' \leq N$ relevant items means that $\frac{N'}{N}$ of recommended items really were relevant, as well as that $\frac{N'}{N}$ of all relevant items really were recommended. This doesn't hold for training set according to Eq. 2.104 where $|target|< N$ for some users. For such users, recall is still from $[0, 1]$, because using N recommendations, it is easily possible to recommended all the $N' < N$ relevant items, but precision will be from $\left[0, \frac{|target|}{N}\right]$ with $\frac{|target|}{N} < 1$ for $|target| < N$. To make both precision and recall be from $[0, 1]$, we can further modify the evaluation procedure by asking for exactly $|target|$ recommendations for each user instead of fixed N across all users.

A completely different methodology of computing $recall@N$ has been proposed in [17]. There is fixed splitting of users into *obs* and *target*, and the training set consists simply of the full set of relevant items for each user:

$$\mathcal{T} = \left\{(u, r^+(u)) \mid u \in U\right\} \quad (2.107)$$

When computing the recall, one user in the testing set is selected (the *active user*). Iteratively, one rated item (the *test item*) is removed from the profile of the active user at time.

The model learned on training set is used to generate Top-N recommendations based on the reduced profile. If the test item is in the Top-N set, a counter is incremented. Given $\mathcal{T}_{\text{val}} \subseteq U \times 2^I$, and a Top-$N$ recommendation model $m: U \times 2^I \to \{I' \subset I \mid |I'| = N\}$ taking both the user and the observation set into account same as in Eqs. 2.105 and 2.106, we can define

$$recall@N_{\text{LOO}}\left(m, \mathcal{T}_{\text{val}}\right) = \frac{|\{(u,i) \mid (u, r^+(u)) \in \mathcal{T}_{\text{val}} \wedge i \in r^+(u) \wedge i \in m\left(u, r^+(u) \setminus \{i\}\right)\}|}{|\{(u,i) \mid (u, r^+(u)) \in \mathcal{T}_{\text{val}} \wedge i \in r^+(u)\}|} \tag{2.108}$$

Such a definition of recall is more practical and fits well to almost any explicit rating dataset, including industrial dataset including users with $|r^+(u)| < N$.

2.5.4.2 Precision and Recall vs. RMSE

Rating prediction models are often used to generate Top-N recommendations such that model $m_{\text{RP}}: U \times I \to \mathbb{R}$ (Eq. 2.8) is provided as the $score: U \times I \to \mathbb{R}$ function (Eq. 2.15) to produce Top-N recommendation model $m_{\text{Top-}N}$ (Eq. 2.11) through mechanism presented in Sec. 2.16. As we mentioned in Sec. 2.5.2.2, popular learning algorithms, such as the MF, are often internally minimizing RMSE to train the model, with RMSE being subsequently also used for its evaluation. It is henceforth natural question how the rating prediction measures like RMSE actually impact Top-N binary classification measures like precision@N and recall@N. In many scenarios, minimizing RMSE effectively only serves as a proxy to maximizing precision@N and recall@N.

Using RMSE as the main criterion in the aftermath of the Netflix Prize Challenge has been questioned e.g. in [55], where using Top-K evaluation approach has been proposed based on ranking. The conclusion was that even "small improvements in RMSE translate into significant improvements in quality of the top K products". However, in a follow-up study [23], where the authors focused on performance evaluation in Top-N recommendation tasks, they denied their general claims in [55] and stated that "the convenient assumption that an error metric such as RMSE can serve as good proxy for top-N accuracy is questionable at best", and that there is no monotonic relation between rating prediction metrics an precision and recall on Top-N recommendation tasks.

Very interesting comparison in RMSE vs. precision and recall was also done in [17] using recall as defined in Eq. 2.108. Measuring relation between RMSE and recall on the MovieLens and the Netflix Prize datasets, the authors even show completely contradictory results: F-measure and recall growing with RMSE, although F-measure and recall express validation reward, whilst RMSE expresses validation loss.

In a bachelor thesis [40] (in Czech) supervised by the author of this thesis, similar results have been achieved with UserKnn following Eq. 2.22 using the *score* function for rating prediction as discussed above.

Our comment to the topic is that nonmonotonicity between RMSE and precision/recall can be trivially shown by considering two rating prediction models $r_{\text{RP}}, r'_{\text{RP}}: U \times I \to \mathbb{R}$ such that $r'_{\text{RP}}(u,i) = r_{\text{RP}}(u,i) + c$ for $c \in \mathbb{R}$. When used for Top-N recommendation, for

different $c \in \mathbb{R}$, we are guaranteed to obtain all the possible values of $\text{RMSE}(m'_{\text{RP}}, \mathcal{T}) \in [\text{RMSE}(m_{\text{RP}}, \mathcal{T}), \infty]$ despite the fact that both models will always have the same precision and recall. But even besides artificial examples like that, results like [17], [23], and [40] show that monotonicity doesn't hold even under "good will" to optimize recall through the RMSE proxy.

2.5.4.3 Popularity-Stratified Recall@N

In [101], it is stated that validation reward measures like precision and recall tend to decrease towards the long tail (less popular items liked by users distinctive taste), while such recommendations are particularly valuable. In response to that, **popularity-stratitified recall@N** is suggested as a reward measure that is biased towards the long tail.

Provided binary rating matrix $\mathbf{R} \in \{0, 1, ?\}^{U \times I}$, we can denote relative item popularity $p\colon I \to [0, 1]$ as

$$p(i) = |\{u \in U \mid r_{u,i} = 1\}| \tag{2.109}$$

Given $\mathcal{T}_{\text{val}} \subseteq \left(U \times 2^I\right) \times 2^I$ as in as in Eqs. 2.103 or 2.104, popularity-stratified recall is given as validation reward function

$$\mathit{recall}^{\beta,w}_{\text{PS}}@N\,(m, \mathcal{T}_{\text{val}}) = \sum_{((u,obs),target)} w^\beta(u) \cdot \frac{\sum_{i \in target \cap m(u,obs)} p(i)^{-\beta}}{\sum_{i \in target} p(i)^{-\beta}} \tag{2.110}$$

where $w^\beta : U \to [0, 1]$ is an optional user-dependent weight, normalized such that $\sum_{u \in U} w^\beta(u) = 1$, with suggested

$$w^\beta(u) = \frac{1}{|U|} \cdot \frac{\sum\limits_{i \in r^+(u)} p(i)^{-\beta}}{\sum\limits_{\substack{v \in U \\ i \in r^+(v)}} p(i)^{-\beta}} \tag{2.111}$$

where $r^+\colon U \to 2^I$ denotes set or relevant items of u as in Eq. 2.103. For $\beta = 0$, we get $\mathit{recall}@N$ as in 2.106. For $\beta = 1$, weight of each item is inverse proportional to its observed popularity. Experiments are done for different $\beta \in [0, 1]$ in [101] with β-weighting not only in evaluation $\left(\text{using } \mathit{recall}^{\beta,w}_{\text{PS}}@N\right)$, but also during training matrix factorization model to optimize for the biased recall. The optimization problem is given for identical to Eq. 2.60 with popularity stratification given by $w_{u,i}$ representing the weight for each (u, i)-part:

$$w_{u,i} = \begin{cases} \dfrac{\sum\limits_{i \in r^+(u)} p(i)^{-\beta}}{\sum\limits_{\substack{v \in U \\ i \in r^+(v)}} p(i)^{-\beta}} & \text{if } \mathit{score}(u, i) \neq ? \\ w_m & \text{if } \mathit{score}(u, i) = ? \end{cases} \tag{2.112}$$

with w_m as a hyperparameter optimized using cross-validation.

2.5.4.4 Catalog and User Coverage

Besides various measures that in general measure the accuracy by comparing $\hat{\mathbf{y}}_\ell = m(\mathbf{x}_\ell)$ with $\mathbf{y}_\ell$ from the validation set (Sec. 2.5.1), multiple alternative measures exist to assess additional qualitative aspects of the provided model. Of the most prominent, the **coverage** measures estimate the portion of users $u \in U$ or items $i \in I$ for which the model is somehow useful.

Specifically, the **catalog coverage** [44, 33, 96] estimates the portion of all the items that provided Top-N recommendation model is able to recommend. Thanks to our general definition of validation reward function in Eq. 2.81, provided $\mathcal{T}_{\text{val}} \subseteq (U \times 2^I) \times 2^I$ and Top-N recommendation model $m\colon U \times 2^I \to \{I' \subset I \mid |I'| = N\}$ as in Eqs. 2.105 and 2.106, we can define catalog coverage as validation reward:

$$\textit{catalog-coverage}\,(m, \mathcal{T}_{\text{val}}) = \frac{1}{|I|}\left|\bigcup_{((u,obs),target)\in\mathcal{T}_{\text{val}}} m(u, obs)\right| \tag{2.113}$$

It is common that different models focus on different subsets of items, leaving the rest unrecommendable. For example, popularity models only recommend items from the top of global popularity list. Popularity model m will hence have $\textit{catalog-coverage}\,(m, \mathcal{T}_{\text{val}})$ very close to N[1]. Collaborative Filtering models, on the other hand, focus only on items with enough ratings, and they do not recommend cold-start items, again focusing them only on a certain subset of items. The same holds for content-based algorithms, such as ItemKnn with attribute similarity (Sec. 2.4.3.2). Such a model is only able to recommend the items with attribute values provided, leaving the unannotated items aside. This makes catalog coverage very meaningful and useful measure of assessing model quality.

There also exists an alternative measure to assess diversity, expressed as **Gini index** [96, 19], which tells how unequally often different items are recommended. Let us define $p(i, \mathcal{T}_{\text{val}})$ as the empirical probability of item $i \in I$ being recommended by Top-N recommendation model m for a random user $u \in U$ in the validation set $\mathcal{T}_{\text{val}}$:

$$p\,(i, \mathcal{T}_{\text{val}}) = \frac{|\{((u, obs), target) \in \mathcal{T}_{\text{val}} \mid i \in m(u, obs)\}|}{\sum_{j\in I} |\{((u, obs), target) \in \mathcal{T}_{\text{val}} \mid j \in m(u, obs)\}|} \tag{2.114}$$

and sequence P of such probabilities for all items $i \in I$ sorted by $p(i)$ in non-decreasing order:

$$P^{\mathcal{T}_{\text{val}}} = \left(p\,(i_1, \mathcal{T}_{\text{val}}), \ldots, p\left(i_{|I|}, \mathcal{T}_{\text{val}}\right)\right) \quad \text{w.r.t. } \forall j, \ell \in \{1, \ldots, |I|\}\colon j < \ell \implies P_j^{\mathcal{T}_{\text{val}}} \leq P_\ell^{\mathcal{T}_{\text{val}}} \tag{2.115}$$

Gini index is then given as a validation reward function:

$$\textit{gini}\,(m, \mathcal{T}_{\text{val}}) = \sum_{j=1}^{|I|} P_j^{\mathcal{T}_{\text{val}}} \cdot (2j - |I| - 1) \tag{2.116}$$

[1] not exactly equal to N but slightly larger because of Eq. 2.12 requirement to not recommend already rated items, and there may be users who have rated items from the top of the list

Yet another alternative diversity measure is the **Shannon entropy** [96], given (using $p(i)$ defined in Eq. 2.114) as validation reward function

$$shannon\left(m, \mathcal{T}_{\text{val}}\right) = -\sum_{i \in I} p\left(i, \mathcal{T}_{\text{val}}\right) \cdot \log\left(p(i, \mathcal{T}_{\text{val}})\right) \tag{2.117}$$

Besides measuring coverage on items, it is also possible to measure coverage on users. The **user coverage** is by far not that popular as catalog coverage, but has been used in several studies, being typically defined [15, 111, 88] as the portion of users from the validation set for which the model was able to generate any recommendation. Formally, it can be defined as a validation reward function

$$\textit{user-coverage}(m, \mathcal{T}_{\text{val}}) = \frac{1}{|U|} \sum_{((u,obs),target) \in \mathcal{T}_{\text{val}}} \begin{cases} 0 & \text{if } m(u, obs) \neq \emptyset \\ 1 & \text{if } m(u, obs) = \emptyset \end{cases} \tag{2.118}$$

An alternative version of user coverage is possible, requiring the model to recommend full set of N requested items:

$$\textit{user-coverage}(m, \mathcal{T}_{\text{val}}) = \frac{1}{|U|} \sum_{((u,obs),target) \in \mathcal{T}_{\text{val}}} \begin{cases} 0 & \text{if } |m(u, obs)| < N \\ 1 & \text{if } |m(u, obs)| = N \end{cases} \tag{2.119}$$

2.5.4.5 Click-Through Rate

All the measures mentioned so far could be classified as the *offline* measures. They are based on split validation (or cross-validation), but they do not require production deployment as the provided training set $\mathcal{T}$ alone is enough for them. As we mentioned in the introduction of Sec. 2.5, there are online measures as well, allowing production evaluation of the deployed model.

One of the most important to mention is the **click-through rate (CTR)** [113, 9, 32]. It assumes a Top-N recommendation model m running in production for a certain period of time, producing set of k recommendations $R = \{r_{u_1,t_1,I_1}, \ldots, r_{u_k,t_k,I_k}\}$ where $I_j \in \{I' \subset I \mid |I'| = N\}$ are the recommended items for the user u_j associated with the j-th recommendation which took place at time t_j. To make the CTR possible to compute, there is also need for user feedback, which is basically a projection $f\colon R \to I \cup \{\varnothing\}$, encoding whether of the recommended items have been clicked by the user in the user interface $\left(f\left(r_{u_j,t_j,I_j}\right) = i \in I_j\right)$ or not $\left(f\left(r_{u_j,t_j,I_j}\right) = \varnothing\right)$. The feedback can either be explicit, collected by proper tracking of clicks in the interface and given as an enumeration $f\colon r_{u_1,t_1,I_1} \mapsto i_1 \quad \ldots \quad r_{u_k,t_k,I_k} \mapsto i_k$, or implicit if proper tracking is not implemented or possible. Implicit feedback can be obtained e.g. by pairing recommendations with future interactions in set $Y = \{(u_1, i_1, t_1, d_1), \ldots, (u_q, i_q, t_q, d_q)\}$ (as introduced in Sec. 2.1.3) such that the recommended item has been interacted within a time window of size $w \in \mathbb{R}^+$ from the recommendation. If more such interactions exists, we take the item from the

interaction with the lowest timestamp fulfilling the time window condition:

$$f\left(r_{u_j,t_j,I_j}\right) = \begin{cases} \left(\underset{\substack{t_\ell > t_j \wedge t_\ell - t_j \leq w \\ i_\ell \in I_j}}{\arg\min_{(u_j,i_\ell,t_\ell,d_\ell)\in R}} t_\ell\right)_2 & \text{if } \exists (u_j, i_\ell, t_\ell, d_\ell) \in R \colon t_\ell > t_j \wedge t_\ell - t_j \leq w \wedge i \in I_j \\ \varnothing & \text{otherwise} \end{cases} \tag{2.120}$$

The CTR of given model is then given simply as:

$$CTR(m) = \frac{|\{r \in R \mid f(r) \neq \varnothing\}|}{|R|} \tag{2.121}$$

CTR is a very simple and is hence quite often in practise when optimizing the models or making business decisions between multiple recommendation system vendors. Online evaluation is typically closely connected with A/B testing [25, 39].

Other, more business and long-term oriented measures exists as well. They include the **Conversion Rate** (CR), **Average Time Spent** (ATS), **Average Order Value** (AOV), and **Return Of Investment** (ROI).

2.5.5 Model Capacity, Underfitting and Overfitting

In Sec. 2.5.1, we defined learning algorithm as a projection which takes the training set $\mathcal{T}_{\text{train}}$ and produces a model. We introduced the notion of split/cross validation as the method of assessing quality of such a model. We stated that with each learning algorithm $\mathcal{A}$, there is a space of hyperparameters $\mathcal{P}_{\mathcal{A}}$ associated, making the process of model construction based not only on provided data, but also on $P \in \mathcal{P}_{\mathcal{A}}$.

Subsequently, we introduced several accuracy measures for different recommendation tasks. Namely the MAE (Eq. 2.88), NMAE (Eq. 2.89), MSE (Eq. 2.90), and RMSE (Eq. 2.91) for Rating Prediction, precision (Eq. 2.94), recall (Eq. 2.95), F-measure (Eq. 2.96), and AUC (Eq. 2.102) for Binary Classification, precision@N (Eq. 2.105), recall@N (Eq. 2.106), leave-one-out recall (Eq. 2.108), and popularity-stratified recall (Eq. 2.110) for Top-N recommendation.

In context of this thesis, it is especially important to mention the relation between accuracy measures and model capacity. The notion of **model capacity** has been introduced in [104] in 1971, with a measure known today as the **Vapnik-Chervonenkis dimension** (or VC dimension for short). It can be simply understood as the maximal size of $\mathcal{T}_{\text{train}}$ such that for arbitrary examples in $\mathcal{T}_{\text{train}}$, it is theoretically possible for learning algorithm $\mathcal{A}$ with given hyperparameterization $P \in \mathcal{P}_{\mathcal{A}}$ to produce a model $m = \mathcal{A}(P, \mathcal{T}_{\text{train}})$ such that $\forall (\mathbf{x}_\ell, \mathbf{y}_\ell) \in \mathcal{T}_{\text{train}} \colon m(\mathbf{x}_\ell) = \mathbf{y}_\ell$, meaning the model captures the whole training set without making an error on it. For such a model $m \colon \mathcal{X} \to \mathcal{Y}$, given accuracy-based validation reward function (such as the recall) $\mathcal{F}$ (as in Eq. 2.78), and given accuracy-based validation loss function (such as the RMSE) $\mathcal{L}$ (as in Eq. 2.81), it holds $\mathcal{F}(m, \mathcal{T}_{\text{train}}) = \max_{m' \in \mathcal{Y}^{\mathcal{X}}} \mathcal{F}(m', \mathcal{T}_{\text{train}})$ and $\mathcal{L}(m, \mathcal{T}_{\text{train}}) = \min_{m' \in \mathcal{Y}^{\mathcal{X}}} \mathcal{L}(m', \mathcal{T}_{\text{train}})$, simply because m, having memorizing it, is optimal of $\mathcal{T}_{\text{train}}$.

Memorizing the whole training set, however, is not always desirable, as the model possibly lacks proper generalization. This is closely connected to property of set of models called the **bias-variance tradeoff**. It refers to a dilemma between trying to simultaneously minimize two sources of model error. The **bias** part of error is caused by abstraction and generalization performed by the learning algorithm while training the model. Due to bias, the resulting model may miss important relations in the data, which is also known as the **underfitting**. The **variance** part of error is caused by sensitivity to noise and learning too granular relations that may be too specific only for the training set, with not enough abstraction, which is also known as the **overfitting**.

As an example, let us consider the UserKnn model on given rating matrix, for simplicity in the unweighted version (Eq. 2.21). In Eq. 2.20, there is a hyperparameter $k \in \mathbb{N}$, determining the number of neighbors to be examined when building the recommendations. Let us further consider that we use this model for rating prediction task putting simply $m(u,i) = \mathit{score}(u,i)$. We may consider that there is a learning algorithm $\mathcal{A}_{\text{UserKnn}}: \mathcal{P}_k \times (\mathbb{R} \cup \{?\})^{U \times I} \to \mathbb{R}^{(U \times I)}$ with $\mathcal{P}_k = \mathbb{N}$ as the single hyperparameter staying for the k. Now consider $m_{k=1} = \mathcal{A}_{\text{UserKnn}}(1, \mathcal{T}_{\text{train}})$ and $m_{k=|\mathcal{T}_{\text{train}}|} = \mathcal{A}_{\text{UserKnn}}(|\mathcal{T}_{\text{train}}|, \mathcal{T}_{\text{train}})$. Ignoring (again for simplicity) possible contradictory examples in $\mathcal{T}_{\text{train}}$, we may say that $m_{k=1}$ has VD dimension equal to ∞, because when predicting for a given user from $\mathcal{T}_{\text{train}}$, the 1st nearest neighbor is the user itself, making the prediction from itself to itself. The algorithm is hence able to fully memorize arbitrarily large set of training examples. Fully memorizing the whole $\mathcal{T}_{\text{train}}$ makes $m_{k=1}$ a low-bias, high-variance model. In contrast, $m_{k=|\mathcal{T}_{\text{train}}|}$ has VC dimension equal to 1, because no matter the source user, all the users are always considered, and because we declared the UserKnn is unweighted, if effectively behaves as a popularity model compliant to Eq. 2.19. Being only able to do global averaging, it effectively behaves as a constant model with high bias and low variance. From these two edge case examples, it should be obvious that some compromise $1 < k < |\mathcal{T}_{\text{train}}|$ is more desirable.

It is important to distinguish between the theoretical model capacity and the actual abilities of the learning algorithm. While in case of UserKnn the learning algorithm has quite easy job just to store the given data into memory for future use, in case of training MF or neural network model, the situation is much more complicated. High-capacity models (large number of latent features, large network) may fail to capture the information during training and get stuck in local optimum e.g. because of wrong learning rate in SGD or in backpropagation. Despite having high VD dimension, such models may exhibit low accuracy even on the training set, caused by both high bias and high variance.

Because manipulation with model capacity is on of the key contributions of this theses, we well revisit the topic in the following chapters.

Chapter 3

Overview of Our Approach

In this chapter, we present our approach of systematically controlling the magnitude of long-tail recommendations generated by the models. The key idea is in picking proper hyperparameters which control the capacity of the resulting model for individual learning algorithms. For each hyperparameter, we discuss its impact on the long-tail recommendations. Besides algorithm-specific hyperparameters, we also propose model-independent popularity-based regularization through generalized β-boosting as proposed in [101], leveraging the unified way in which we introduced the algorithms in the previous chapter. Finally, we propose searching for Pareto-optimal front of models in multiobjective top-N evaluation, proposing specifically the leave-one-out recall and the catalog coverage as the most relevant measures for optimizing long-tail recommendations.

3.1 Popularity-Based Regularization

In Sec. 2.4, we introduces several recommendation algorithms producing models that can be used for different recommendation tasks: Rating Prediction, Binary Classification, Top-N Recommendation, and Ranking. The resulting models were described under unified framework of the *score function score*: $U \times I \to \mathbb{R}$ (Eq. 2.15), showing that provided such a function, we can easily construct a recommendation model for any of the aforementioned tasks.

In Sec. 2.5.4.3, we discussed an approach from [101], where a novel method for model evaluation bonifying long-tail recommendations, coming from assumption that such recommendations are particularly valuable. Together with the proposed evaluation method (the popularity stratified recall), a biased version of training the MF model has been proposed in [101], showing that model trained using popularity stratified weighting in the loss function leads to model biased towards the long tail. Specifically, hyperparameter $\beta \in [0, 1]$ in proposed to control the size of popularity stratified bias, with $\beta = 0$ meaning zero bias, and $\beta = 1$ full bias.

Based on long-term experience in building and running productionalized models, in-

cluding empirical, offline, and online evaluation, we independently discovered the very same hyperparameter, finding it very important for model quality. While in [101] the hyperparameter is used in the loss function of MF learning algorithm, putting higher weight observed ratings $r_{u,i}$ where $i \in I$ is a long-tail item, we propose large generalization to this approach.

Our generalization includes:

1. Excluding β-biasing from the MF model training (loss function in Eq. 2.112) and postponing it for the recommendation phase.

2. Making β-biasing applicable to **any** of the presented models. To do that, we exploit the fact that all the models were presented using the internal $score\colon U \times I$ function. Given such a function, and a general rating matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{U \times I}$, we define the **popularity-stratified score** as follows:

$$score^{\beta}_{\mathrm{PS}}(u,i) = \begin{cases} \dfrac{score(u,i)}{|\{v \in U \mid r_{v,i} > 0\}|^{\beta}} & \text{if } \exists v \in U\colon r_{v,i} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

   This makes β-biasing applicable not only for the original version of MF (Eq. 2.60 with Eq. 2.112), but also for:

   - UserKnn
     - unweighted as in Eq. 2.21,
     - weighted as in Eq. 2.22,
     - with non-normalized neighborhood following [23] as in Eq. 2.23,
   - ItemKnn as introduced in [93]
     - unweighted as in Eq. 2.29,
     - weighted as in Eq. 2.30,
     - with non-normalized neighborhood following [23] as in Eq. 2.31,
   - Association Rules
     - using on the best-rule method based on [91] and [61] as in Eq. 2.44,
     - using weighted voting based on idea from [54] as in Eq. 2.45,
     - using different rule-quality measures based on [6] as in Eqs. 2.41, 2.42, and 2.43,
   - Matrix Factorization
     - in simplest form as in 2.46,
     - with user, item, and global biases following [56] and Eq. 2.50,
     - with mixed implicit and explicit data proposed in [55] and Eq. 2.52,
     - with imputed unobserved values following [100] and Eq. 2.59,

  - using different learning objectives in model learning phase: Eq. 2.47, Eq. 2.51 [56], Eq. 2.53 [55], Eq. 2.56 [47],
  - trained using different optimization algorithms (SGD, ALS),
- Deep Learning Models
  - Wide & Deep Learning proposed in [20] as in Eq. 2.71,
  - U-AutoRec and I-Autorec proposed in [95] as in Eqs. 2.72 and 2.73.

3. Generalizing β-boosting also for $\beta < 0$. In [101], the assumption is that the (MF) model generally tends to be biased towards popular items, and hence correction towards the long-tail is needed using $\beta \in [0, 1]$. We state that this is not always the case and that there are models biased towards long-tail in default behavior. For such models, $\beta < 0$ makes sense in order to *bonify* popular items instead on *penalizing* them using $\beta > 0$. In general, we propose $\beta \in \mathbb{R}$ as a universal model hyperparameter with $\beta \in [-1, 1]$ being the most reasonable in most cases.

3.2 Manipulating Model Capacity

In Sec. 2.5.5, we introduced the notion of the bias-variance trade-off connected with model capacity. We also mentioned that the capacity of the resulting model can be often controlled by the hyperparameterization of the learning algorithm (as introduced in Sec. 2.2.1). One the most important contributions of this thesis is showing that all the models presented in Sec. 2.4 have a learning algorithm $\mathcal{A}$ with hyperparameterization $\mathcal{P}_\mathcal{A}$ that directly affects the capacity of the model, leading to similar behavior for all the models in the Recall-Coverage plane.

Therefore, in this section, we will introduce such hyperparameterization for different recommendation algorithms from Sec. 2.4. Assuming $\mathcal{A}$ is a learning algorithm hyperparameterized by $\mathcal{P}_\mathcal{A}$, let us denote $\mathcal{P}_\mathcal{A} = \{p_1 \in D_1, \ldots, p_r \in D_r\}$ the fact hyperparameter space $\mathcal{P}_\mathcal{A}$ is a cartesian product $D_1 \times \ldots \times D_r$ with individual hyperparameters $p_1 \in D_1, \ldots, p_r \in D_r$. Furthermore, let us denote $[p_1 = d_1, \ldots, p_r = d_r] \in \mathcal{P}_\mathcal{A}$ the one specific hyperparameterization given by value assignments to individual hyperparameters. Let us discuss capacity hyperparameters for individual models in the following subsections.

3.2.1 β as Universal Model Capacity Hyperparameter

In [101], the β-stratification has been introduced on top of strong assumption that the missing data (denoted as $r_{u,i} = ?$ throughout this thesis) are not distributed in the rating matrix at random, which is called the MNAR (Missing Not At Random) assumption. It is stated that there is positive correlation between item popularity $p(i)$ of item $i \in I$ and the probability of observing the rating for the item $\Pr(r_{u,i} \neq ? \mid i)$, which is actually against true users' interests. It is furthermore stated that such as a bias is by default learned by the MF model, suggesting popularity-based weighting of the interactions in the rating matrix

with $\beta = 0$ corresponding to no correction, and $\beta = 0.5$ as the correction to situation when $\Pr(r_{u,i} \neq ? \mid i)$ grows linearly with $p(i)$.

While we don't generally agree that all the models by default capture the MNAR trend (and we will show counterexamples in the experiments), we agree thar MNAR is valid in general. It often happens in many domains that some items are generally well-known (for example, thanks to good marketing), making users search for them and hence view them much more likely simply because they are popular. Another source of MNAR bias may be influencing users by editorial choice, such as putting hand-picked items on the home/welcome page of the system. A good example is online news, where articles about recent important events are put on the frontpage.

Let us consider usage of β as proposed in Eq. 3.1 on top of the *score* function of an arbitrary model. We defined the $score(u, i)$ as the relevance of item $i \in I$ to user $u \in U$.

Now consider a **low-capacity** model m based on *score*. As discussed in Sec. 2.5.5, m has high bias and low variance. The learning algorithm is hence only able to capture general trends in data into m, being unable to encompass rare events. Because of the MNAR assumption, the general trend is that popular items are often relevant, resulting in high value of $score(u, i)$ for popular items i among the all users $u \in U$. Simply said, popular items have high score for everybody in such weak learners. As an example, imagine a product which is very popular in a supermarket, such as the shopping bag. Based on the purchase data, shopping bag seems like a very good recommendation for everybody: no matter what is being purchased, it is put to the shopping bag in the end. If we take, for example, ARs with *confidence* (Eq. 2.41) as the rule quality measure, there will be lot of rules with shopping bag on right-hand side with confidence close to highest possible value of 1. That is because 99% of users who purchase diapers also purchase a shopping bag, as well as 99% of user who purchase beer, purchase shopping bag, too.

When we put $\beta > 0$, we start correcting that: Items $i \in I$ that are globally not so popular, but are still identified as possibly slightly relevant for given user u, that is $score(u, i) > 0$, start to be released from their unpopularity handicap, and possibly replace the original popular items in, say, Top-N recommendation. In our example with ARs and *confidence* quality measure, some other items with lower confidence will replace the shopping bag, hopefully recommending goods that are more relevant, such as the baby oil for user with diapers in the shopping cart, despite $conf(\{diapers\} \implies \{baby\text{-}oil\}) = 0.2$, unlike $conf(\{diapers\} \implies \{shopping\text{-}bag\}) = 0.99$.

In contrast, consider a **high-capacity** model m based on *score*. Such a model has high variance and low bias. A good example may be the UserKnn model with small k. The model tends to capture even the small user groups and offer niche recommendations to them. The problem here, however, may be that the model is too long-tail. Imagine an online news portal with a niche spare time section about cycling, and an anonymous user who came to that portal and viewed one or two cycling articles. The most similar users will also have cycling articles in their history. Henceforth, there is very high chance that the recommended article will be about cycling again, simply because too small cluster of niche users has been considered. Viewing an article about cycling, however, doesn't mean that general news are irrelevant for that user, because many articles are by their

nature interesting to everybody. In this case, the error doesn't come from high bias as in the "shopping bag example", but, quite oppositely, from the high variance. In this case, $\beta < 0$ is the correction. Because articles about cycling are popular only among small set of users, but unpopular in general, they will start being penalized. This may cause that if some of the neighboring users also viewed something more mainstream, it will get boosted and replace the long-tail items. Instead of another cycling article, the user may be recommended some generally popular article about sports, releasing her from the "cycling bubble".

In general, each model has some default bias-variance ratio, and using negative or positive β, we can shift it in one or the other direction. For extreme values of $\beta \to -\infty$ and $\beta \to \infty$, we may obtain the popularity and "antipopularity" models, respectively, at least on the set of items $\{i \in I \mid score(u, i) \neq 0\}$ for given $u \in U$. In the our experiments, we will demonstrate the impact of β on different model.

3.2.2 User-Based k-Nearest Neighbors

For the UserKnn learning algorithm $\mathcal{A}_{\text{UserKnn}}$, the space of capacity-manipulating hyperparameters is

$$\mathcal{P}_{\mathcal{A}_{\text{UserKnn}}} = [k \in \mathbb{N}, \beta \in \mathbb{R}] \tag{3.2}$$

We basically discussed impact of both hyperparameters in the previous sections while using UserKnn as example to introduce model capacity (Sec. 2.5.5) and universal β-boosting (Sec. 3.2.1). Just to summarize: small value of k leads to low-bias, high-variance model, which is prone to overfitting. If the k is extremely small, such as $k = 1$, the model may capture pure noise. Oppositely, for high value of k, we obtain a high-bias, low-variance model. Especially for the unweighted version (Eq. 2.21), such a model is prone to overfitting, yielding pure popularity model (Eq. 2.19) for $k \to \infty$.

3.2.3 Item-Based k-Nearest Neighbors

The ItemKnn learning algorithm $\mathcal{A}_{\text{ItemKnn}}$ has, similarly to $\mathcal{A}_{\text{UserKnn}}$, capacity-manipulating hyperparameterization

$$\mathcal{P}_{\mathcal{A}_{\text{ItemKnn}}} = [k \in \mathbb{N}, \beta \in \mathbb{R}] \tag{3.3}$$

In case of ItemKnn, however, the k has a different meaning, and hence deserves separate discussion. The difference can be best illustrated on a user $u \in U$ who had rated only single item i_u, that is $\{i \in I \mid r_{u,i} \neq ?\} = \{i_u\}$. For such a user, considering the Top-N recommendation task, the k doesn't have significant impact, because the candidates are sorted according to the *sim* function in Eq. 2.27, and only the top k most similar can obtain non-zero score (they appear in $\text{NN}_k(i_u)$ in Eq. 2.28 and hence also in Eqs. 2.29, 2.30, or 2.31, depending on the ItemKnn version used). Specifically, for Top-N recommendation, it is always the Top-$(\min(N, k))$ items most similar to i_u which are recommended for such a single-item user u. With $N = 10$, the recommendations are all the same $\forall k \in \{10, 11, \ldots\}$.

This would not be the case in UserKnn where the recommendations for such user can change easily with growing k.

Having mentioned that, it is the users with more that 1 rated item, for who bigger k plays a role, because multiple neighborhoods get mixed together. Provided well-working *sim* function, ItemKnn is quite good long-tail recommender by default. The rating-based cosine similarity as proposed in the original paper [93], works quite well. It tends to recommend long-tail items to long-tail items, because the similarity is basically measured by the intersection of sets users who interacted with both. Let us consider the "diapers" example from Sec. 3.2.1, and for simplicity, only binary rating matrix, such the one created from purchases. Furthermore, consider set $D \subseteq U$ of users who purchased diapers. The set of users $O \subseteq U$ who purchased baby oil will be much more similar to D compared to set of users $B \subseteq U$ who purchased the shopping bag. Even though $|D \cup O|$ may be smaller than $|D \cup B|$, it is the very large $|B|$ which highly penalizes B in the cosine similarity reduced to Ochiai similarity coefficient $\frac{|D \cap B|}{|D| \cdot |B|}$ (because the rating matrix is binary).

3.2.4 Association Rules

In case of Association Rules, the capacity-manipulating hyperparameterization $\mathcal{P}_{\mathcal{A}_{\text{AR}}}$ of the learning algorithm $\mathcal{A}_{\text{AR}}$ is:

$$\mathcal{P}_{\mathcal{A}_{\text{AR}}} = [s_{\min} \in [0, 1], \beta \in \mathbb{R}] \tag{3.4}$$

In Eq. 2.38, we defined $s_{\min}$ as the minimal popularity (relative to $|U|$) of an itemset $X \cup Y$ for an association rule $X \Rightarrow Y$. Therefore, it directly controls the granularity of the discovered rules, and hence the capacity of the model.

For high value of $s_{\min}$, the ARs model m has low capacity, and hence exhibits high bias and low variance. Only the most important associations are learned in such case, because for rule $X \Rightarrow Y$, there must be at least $\lceil s_{\min} \cdot |U| \rceil$ users $u \in U$ having $r_{u,i} = 1 \; \forall i \in X \cup Y$ in the given binary rating matrix $\mathbf{R} \in \{0, 1, ?\}^{U \times I}$.

For small value of $s_{\min}$, on the other hand, the resulting model has high capacity, leading to low bias and high variance. It is able to capture rules also for the long-tail items, not requiring too many users to support a rule. In an extreme case of $s_{\min} \to 0$, there will be an explosion in the number of rules as the learning algorithm will extract rules as granular as supported by only single user. That is, for user $u \in U$ who positively rated items $r^+(u) = \{i \in I \mid r_{u,i} = 1\}$, we are guaranteed that all the rules $\{X \Rightarrow Y \mid X \subset r^+(u) \wedge Y \subset r^+(u)\}$ will be contained in the resulting model. This is an obvious overfitting.

In case of ARs, it is also the β hyperparameter which deserves separate discussion. In Sec. 2.4.4.3, we mentioned that the original paper proposing the *best-rule* method [91] came with the *confidence* as the suggested rule-quality measure. Let us consider special class of rules $X \Rightarrow Y$ where the right-hand side contains only one item, that is $Y = \{i_y\}$. Ignoring the cases when no rules are applicable, assuming rules with $|Y| = 1$, and using confidence (Eq. 2.41) as the rule-quality measure, the *score* function for the *best-rule* method from

Eq. 2.44 simplifies to

$$score(u,i) = \max_{\substack{(X \Rightarrow \{i\}) \in \mathcal{R} \\ T(u) \subseteq X}} conf(X \Rightarrow \{i\}) \tag{3.5}$$

and for the *weighted voting* method from Eq. 2.45 to

$$score(u,i) = \sum_{\substack{(X \Rightarrow \{i\}) \in \mathcal{R} \\ T(u) \subseteq X}} conf(X \Rightarrow \{i\}) \tag{3.6}$$

Looking once more at our definition of general β-boosting in Eq. 3.1 and considering our definition of support in Eq. 2.37, it appears that in case of binary matrices, $score^{\beta}_{\mathrm{PS}}$ rewrites to nothing but

$$score^{\beta}_{\mathrm{PS}}(u,i) = \frac{score(u,i)}{supp\left(\{i\}\right)^{\beta}} \tag{3.7}$$

Applied to Eq. 3.5:

$$score^{\beta}_{\mathrm{PS}}(u,i) = \frac{\max\limits_{\substack{(X \Rightarrow \{i\}) \in \mathcal{R} \\ T(u) \subseteq X}} conf(X \Rightarrow \{i\})}{supp\left(\{i\}\right)^{\beta}} = \max_{\substack{(X \Rightarrow \{i\}) \in \mathcal{R} \\ T(u) \subseteq X}} \frac{conf(X \Rightarrow \{i\})}{supp\left(\{i\}\right)^{\beta}} \tag{3.8}$$

Applied to Eq. 3.6:

$$score^{\beta}_{\mathrm{PS}}(u,i) = \frac{\sum_{\substack{(X \Rightarrow \{i\}) \in \mathcal{R} \\ T(u) \subseteq X}} conf(X \Rightarrow \{i\})}{supp\left(\{i\}\right)^{\beta}} = \sum_{\substack{(X \Rightarrow \{i\}) \in \mathcal{R} \\ T(u) \subseteq X}} \frac{conf(X \Rightarrow \{i\})}{supp\left(\{i\}\right)^{\beta}} \tag{3.9}$$

In both cases, β propagated to the rule-quality measure, replacing $conf\left(X \Rightarrow \{i\}\right)$ with

$$\frac{conf(X \Rightarrow \{i\})}{supp\left(\{i\}\right)^{\beta}} \tag{3.10}$$

Now putting β=0, we get the original $conf\left(X \Rightarrow \{i\}\right)$, which is not surprising. But putting $\beta = 1$, we get $\frac{conf(X \Rightarrow \{i\})}{supp(\{i\})}$, which is nothing but the $lift(X \Rightarrow \{i\})$ according to Eq. 2.42. In case of ARs, $\beta \in [0,1]$ can be hence thought as hyperparameter for continuous transition from confidence to lift, if confidence was provided as the original measure. Similarly, for $\beta \in [-1,0]$, we get continuous transition from lift to confidence, if lift was provided as the original measure. While the confidence tends to be biased towards globally popular items, the lift is a long-tail recommender.

3.2.5 Matrix Factorization

For Matrix Factorization learning algorithms, there are multiple capacity hyperparameters:

$$\mathcal{P}_{\mathcal{A}_{\text{MF}}} = [f \in \mathbb{N}, \lambda \in \mathbb{R}, \beta \in \mathbb{R}] \tag{3.11}$$

The number of factors f is a natural capacity hyperparameter simply from the logic that the larger matrices $\mathbf{P} \in \mathbb{R}^{f \times U}$ and $\mathbf{Q} \in \mathbb{R}^{f \times I}$, the more data from the training set can be stored into them. For very large f, in extreme case $f = \max(|U|, |I|)$, it is possible for the learning algorithm to learn $\mathbf{P}$ and $\mathbf{Q}$ on arbitrary rating matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{U \times I}$ such that $\mathbf{P}^T \cdot \mathbf{Q}$ reconstructs the matrix without error, that is $\forall (u,i) \in U \times I : r_{u,i} \neq ? \implies \mathbf{p}_{*,u}^T \cdot \mathbf{q}_{*,i} = r_{u,i}$. Such a model for high f hence represents low-bias, high-variance model prone to overfitting with no abstraction learned from the original rating matrix.

In contrast, lower values of f push the learning algorithm to generalize, producing high-bias, low-variance model. In extreme case of $f = 1$, the model tends to behave like a popularity model. For $f = 1$, instead of being represented by a vector of factors, each user $u \in U$ and each item $i \in I$ are represented by a single number, $p_u \in \mathbb{R}$ and $q_i \in \mathbb{R}$, respectively. In order to minimize the objective function of the optimization problem (Eqs. 2.47, 2.51, 2.53, or 2.56), the learning algorithm is forced to assign the highest values to those p_us of the highest $\sum_{\substack{i \in I \\ r_{u,i} \neq ?}} r_{u,i}$, and to those q_is of the highest $\sum_{\substack{u \in U \\ r_{u,i} \neq ?}} r_{u,i}$. When such a model is about to recommend to user $u \in U$, then as long as $p_u > 0$, it holds $score(u,i) \propto q_i$, and hence the popularity model behavior in Top-N recommendation.

Considering the hyperparameter $\lambda \in \mathbb{R}_0^+$, it is intended as a regularization, which makes limiting model capacity its primary purpose. For small values of λ, we get low-bias, high-variance model. For $\lambda = 0$, the learning algorithm may minimize the objective function by using very large numbers in the factor matrices, achieving perfect fit the observed values, but possibly completely out-of-scale results when computing $\mathbf{p}_{*,u}^T \cdot \mathbf{q}_{*,i}$ for unobserved $r_{u,i} = ?$. This can be seen as clear overfitting.

For $\lambda \gg 0$, on the other hand, the objective function prevents the learning algorithm to fit the observed values, because the losses from high $\|\mathbf{p}_{*,u}\|^2$ and $\|\mathbf{q}_{*,i}\|^2$ start getting higher importance than $\sum_{\substack{(u,i) \in U \times I \\ r_{u,i} \neq ?}} (r_{u,i} - \mathbf{p}_{*,u}^T \cdot \mathbf{q}_{*,i})^2$, leading to low-capacity model.

3.2.6 AutoRec

The last model we will briefly discuss is the AutoRec. For both the U-AutoRec and I-AutoRec, the capacity hyperparameterizations are the same:

$$\begin{aligned} \mathcal{P}_{\mathcal{A}_{\text{U-AutoRec}}} &= [f \in \mathbb{N}, \lambda \in \mathbb{R}, \beta \in \mathbb{R}] \\ \mathcal{P}_{\mathcal{A}_{\text{I-AutoRec}}} &= [f \in \mathbb{N}, \lambda \in \mathbb{R}, \beta \in \mathbb{R}] \end{aligned} \tag{3.12}$$

In Sec. 2.4.6.2, Eqs. 2.76 and 2.77, we showed that both the U-AutoRec and I-AutoRec can be thought as non-linear generalizations of MF. Henceforth, the roles of $f \in \mathbb{N}$ and $\lambda \in \mathbb{R}$ remain basically the same as discussed in the previous Sec. 3.2.5.

3.3 Multi-Objective Top-N Evaluation

In Sec. 2.2.1, we introduced a formal framework for hyperparameterized learning algorithms that can produce models on given training dataset/rating matrix. After the introduction of various algorithms in Sec. 2.4, in Sec. 2.5.1 we also introduced a formal framework for evaluation of both the individual models and the learning algorithms that produce them. In the same section, we introduced various performance measures.

Our approach lies in putting the components together, focusing namely on the **Top-N Recommendation task** as defined in Sec. 2.2.4. The reasons why we consider this task the most important have been mentioned at the beginning of Sec. 2.2.4: It the most important, practical, and widespread task in industrial practice. While the other tasks and approaches to their evaluation focus also on (u, i) pairs other than those where user $u \in U$ will actually **see** the item $i \in I$, Top-N Recommendation focuses exactly on those N items from i, that may bring value to the user or the service provider: the items which that **are presented in the user interface**. We believe that evaluation measures build on top of these N items are henceforth the most important, because all the remaining items are practically irrelevant.

3.3.1 Searching For Pareto-Optimal Models

We showed that various performance measures exist for the Top-N recommendations task. We believe that considering only single measure is insufficient for practical evaluation, and that searching for good models based on multiple criteria is the right approach.

For the purposes of multi-objective optimization, it is useful to define *Pareto-domination* and *Pareto-optimality* first. Let X be a set and let $F = \{f_1, \ldots, f_n\} \subseteq \mathbb{R}^X$, that is $f_i \colon X \to \mathbb{R} \quad \forall i \in \{1, \ldots, n\}$. We say that $x \in X$ **Pareto-dominates** $x' \in X$ across F, denoting $x \succ_F x'$, if and only if the following two conditions are satisfied:

1. $\forall i \in \{1, \ldots, n\} \colon f_i(x) \geq f_i(x')$,
2. $\exists i \in \{1, \ldots, n\} \colon f_i(x) > f_i(x')$.

We also say that $x^* \in X$ is **Pareto-optimal** across F, denoting $\mathrm{PO}_F(x^*)$, if and only if it is *not* Pareto-dominated by any $x \in X$, that is $\mathrm{PO}_F(x^*) \iff \neg \exists x \in X \colon x \succ_F x^*$. Finally, we define **Pareto-optimal front** across F, denoted $\mathrm{PF}(F)$, as the set of all Pareto-optimal elements across F, that is $\mathrm{PF}(F) = \{x^* \in X \mid \mathrm{PO}_F(x^*)\}$. The graphical intuition of Pareto-optimal front for $F = \{f_1, f_2\}$ and set X such that $\{(f_1(x), f_2(x)) \mid x \in X\}$ form 3 continuous curves in $\mathbb{R}^2$, is presented in Fig. 3.1.

In our context, the functions from F are the validation reward functions for individual models as defined in Eq. 2.81, with various instances for Top-N Recommendation task introduced in Sec. 2.5.4. Given training dataset $\mathcal{T}$ partitioned into $\mathcal{T}_{\text{train}}$ and $\mathcal{T}_{\text{val}}$ as in Sec. 2.2.1, set of validation reward functions $F = \{\mathcal{F}_1, \ldots, \mathcal{F}_n\}$, and set of hyperparameterizable Top-N learning algorithms $\mathfrak{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_r\}$, we may put

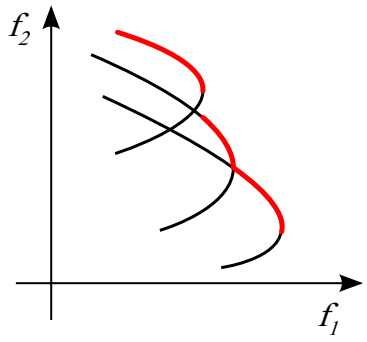

Figure 3.1: Visual intuition of Pareto-optimal front.

$X = \{\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right) \mid \mathcal{A} \in \mathfrak{A} \wedge P_{\mathcal{A}} \in \mathcal{P}_{\mathcal{A}}\}$. We may then say that we are basically searching for all the algorithms tied with their hyperparameterization, $(\mathcal{A}, P_{\mathcal{A}})^*$, such that $\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right)$ produces a model belonging to Pareto-optimal front across F, which is generalization of statement from Sec. 2.5.1. Formally:

1. For $\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right) \in X$ and $\mathcal{A}'\left(P'_{\mathcal{A}'}, \mathcal{T}_{\text{train}}\right) \in X$, it holds $\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right) \succ_F \mathcal{A}'\left(P'_{\mathcal{A}'}, \mathcal{T}_{\text{train}}\right)$ if and only if the following two conditions are met:

   a) $\forall \mathcal{F}_i \in F\colon \mathcal{F}_i\left(\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right), \mathcal{T}_{\text{val}}\right) \geq \mathcal{F}_i\left(\mathcal{A}'\left(P'_{\mathcal{A}'}, \mathcal{T}_{\text{train}}\right), \mathcal{T}_{\text{val}}\right)$,

   b) $\exists \mathcal{F}_i \in F\colon \mathcal{F}_i\left(\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right), \mathcal{T}_{\text{val}}\right) > \mathcal{F}_i\left(\mathcal{A}'\left(P'_{\mathcal{A}'}, \mathcal{T}_{\text{train}}\right), \mathcal{T}_{\text{val}}\right)$,

2. $\text{PO}_F\left(\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right)^*\right) \iff \neg\exists \mathcal{A}'\left(P'_{\mathcal{A}'}, \mathcal{T}_{\text{train}}\right) \in X\colon \mathcal{A}'\left(P'_{\mathcal{A}'}, \mathcal{T}_{\text{train}}\right) \succ_F \mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right)^*$,

3. $\text{PF}(F) = \{\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right)^* \in X \mid \text{PO}_F\left(\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right)^*\right)\}$.

Please note that for simplicity, we assume that $\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right)$ if a fixed model as a result of one particular model training. This holds also for Sec. 2.5.1 where the validation is meant as practical. Assuming that model learning is non-deterministic in general and may produce different models over multiple runs, we could further generalize our definitions, replacing $\mathcal{F}_i\left(\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right), \mathcal{T}_{\text{val}}\right)$ with $\mathbb{E}\mathcal{F}_i\left(\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right), \mathcal{T}_{\text{val}}\right)$ as the expected value of $\mathcal{F}_i\left(\mathcal{A}\left(P_{\mathcal{A}}, \mathcal{T}_{\text{train}}\right), \mathcal{T}_{\text{val}}\right)$ over infinite number of runs. We don't use such generalization in this thesis, simply assuming that running k-fold cross-validation, with k single training iterations as in Eq. 2.86, is sufficient to get stable enough and computationally feasible estimate.

3.3.2 Recall-Coverage Optimization

Because one of the main contributions of this thesis is manipulation with model capacity as stated in Sec. 3.2, we propose the following two Top-N validation reward functions as the most important for multi-objective Top-N evaluation:

- *recall*@N_{LOO} as defined in Eq. 2.108, **extended by normalization**,

- *catalog-coverage* as defined in Eq. 2.113.

In our experiments, we will show that using $F = \{recall@N_{\text{LOO}}, catalog\text{-}coverage\}$ makes very good sense when searching for Pareto-optimal models, because both the measures:

- strongly correlate with model capacity,
- are contradictory when manipulating model capacity.

Let us explain our modification of $recall@N_{\text{LOO}}$, which adds user-based normalization. In its original version, proposed in [17] and shown in Eq. 2.108, there is a "global counter" for all items hidden from the profiles of all the users. From a higher perspective, this is somehow similar to standard versions of Rating Prediction measures, such as the MAE or the RMSE, as well – they also average across all hidden ratings, ignoring their source users. The resulting yet questionable behavior of accuracy measures implemented in this way is that the more hidden ratings from given user $u \in U$, the more this user impacts the value of the measure. It may easily happen that for two users $u, u' \in U$, there is a huge difference in number of observed ratings in given rating matrix $\mathbf{R}$, that is $|\{i \in I \mid r_{u,i} \neq ?\}| \ll |\{i \in I \mid r_{u',i} \neq ?\}|$. Considering the original leave-one-out recall, $recall@N_{\text{LOO}}$, users will impact the measure proportionally to their $|r^+(u)|$.

While such behavior may be acceptable on clean, well-preprocessed academic datasets, in industrial datasets, such disproportions in $|r^+(u)|$ may cause serious issues. Considering, for example, implicit rating matrices generated from clickstreams on Internet websites, there are many users with only very few interactions (see the $|r^+(u)|$ distributions on industrial datasets used in our experiments in Table 4.2). It does not hold that such users are unimportant from business perspective: they may be newcomers acquired from a search engine, and it may be desirable to provide them quality recommendations as fast as possible to keep them on the site. In contrast, if the data isn't properly cleaned, there may be users with thousands or even hundreds of thousands of interactions, representing robots and crawlers. It does not hold that these users are in order of magnitude more important. Instead, they behave rather randomly and are irrelevant for the evaluation.

To address these issues, we propose **user-normalized leave-one-out recall@N** validation reward function as a modification to Eq. 2.108 given as:

$$recall@N^{\text{UN}}_{\text{LOO}}\left(m, \mathcal{T}_{\text{val}}\right) = \frac{1}{|T_{\text{val}}|} \sum_{(u, r^+(u)) \in \mathcal{T}_{\text{val}}} \frac{|\{i \in r^+(u) \mid i \in m\left(u, r^+(u) \setminus \{i\}\right)\}|}{|r^+(u)|} \tag{3.13}$$

With $recall@N^{\text{UN}}_{\text{LOO}}$, each user's leave-one-out recall is normalized to $[0, 1]$, and the final value is computed as the average across all users. As a result, each user in the validation set contributes to the recall with equal weight. The measure could be further improved we user-dependent weighting, but for our experiments, we consider $recall@N^{\text{UN}}_{\text{LOO}}$ sufficient.

As a complementary measure to $recall@N^{\text{UN}}_{\text{LOO}}$, we use *catalog-coverage*, because it well indicates the long-tailness of the recommendation. Being relatively simple compared to

more sophisticated measures such as the Gini index (Eq. 2.116) or Shannon entropy (Eq. 2.116), it gives decent yet easy to interpret insight into how the recommendations vary for different users. It is expected that low-capacity models will rather small catalog coverage, because they can learn relations only for the most popular items, as has been discussed earlier. In contrast, high-capacity models with long-tail recommendation will have larger catalog coverage, because they are able to capture unique taste of long-tail users, and hence recommend long-tail items to them. We will support this claim with experimental results in Chap. 4.

We call our approach the **Recall-Coverage Optimization (RCO)**. In Sec. 2.5.4.3, we also mentioned the *popularity-stratified Recall@N* as proposed in [101], which is the same paper where β-based popularity stratification has been introduced. The author uses β-stratified recall $recall^{\beta,w}_{\mathrm{PS}}@N$ (Eq. 2.110) as a measure capturing both the long-tailness and the recall into single number. Although we recognize the approach as very smart and logically motivated, in our opinion, a significant disadvantage is that $recall^{\beta,w}_{\mathrm{PS}}@N$ takes β as a parameter, which may lead to concurrent optimization of the model and the measure itself. As stated by the author himself, the exponent β used in model training can be expected to be similar in value to β used in $recall^{\beta,w}_{\mathrm{PS}}@N$. That is rather logical, because for chosen β in the measure, similar β will be optimal for the model. In our approach of multi-objective optimization, two well-defined and fixed measures are used, allowing us to explore model behavior based on algorithm hyperparameterization in a convenient and systematic way.

Chapter 4

Main Results

In this chapter, we present number of experiments showing that our concepts proposed in the previous chapter really work as expected. Specifically, we perform exhaustive recall-coverage evaluation for all the previously proposed capacity-controlling hyperparameters of different learning algorithms. The experiments are done on a mixture of publicly available and industrial dataset, showing the robustness of the proposed methods and stability of the results.

4.1 Experimental Setup

Let us first introduce the datasets we selected for our experiments, as well as the selected learning algorithms with brief implementational notes.

4.1.1 Experimental Datasets

For our experiments, we have chosen mixture of publicly available datasets and industrial datasets. A high-level overview of the algorithms is shown in Table 4.1. While the first three (MovieLens 1M, MovieLens 20M, and Last.FM 2K) are publicly available, the rest (BUBB.Store, Casa Cenina, Just Spotted, Moodings) are closed, industrial datasets.

Tables 4.2 and 4.3 offer an insight to the distribution of numbers of interactions per user and per item, respectively. P_1 stays for the 1st percentile, Q_1 is the first quartile (25th percentile), Q_2 is the median (50th percentile), Q_3 is the third quartile (75th percentile) and P_{99} is the 99th percentile.

As can be seen especially from Table 4.2, there is huge difference between public and industrial datasets. Because we intentionally didn't do any cleaning such as removing users with too few interactions, a trend common for online websites is obvious: Vast majority users come from ephemeral, anonymous sessions, without the possibility to collect long-term interactions from them. This corresponds to practical conditions, where the recommendation algorithms must be able to respond to newly incoming data and generate as quality recommendations as possible from the data collected so far.

The following subsections provide brief descriptions of individual datasets.

Table 4.1: Interaction Types and Item Attributes of Experimental Datasets

Dataset Name	**Total Interactions**	**Interaction Types**	**Item Attributes**
MovieLens 1M	1,000,209	Explicit ratings	Genres, Title, Year
MovieLens 20M	20,000,263	Explicit ratings	Genres, Title, Year
Last.FM 2K	92,834	Plays	$\emptyset$
BUBB.Store	8,697,556	Detail-views, Cart-additions, Purchases, Explicit ratings	Store ID, Brand, Categories, Name
Casa Cenina	9,891,560	Detail-views, Bookmarks, Cart-additions, Purchases, Explicit ratings	Categories, Product tags, Manufacturer, Product model
Just Spotted	875,013	Detail-views, Bookmarks, Cart-additions, Purchases, Explicit ratings	Product name, Vendor, Type
Moodings	2,884,636	Detail-views, Bookmarks, Cart-additions, Purchases, Explicit ratings	Product name, Vendor, Type

4.1.1.1 MovieLens 1M and MovieLens 20M Datasets

These are very well known datasets collected over years by the GroupLens Research laboratory in the Department of Computer Science and Engineering at the University of Min-

Table 4.2: User Interaction Characteristics of Experimental Datasets

Dataset Name	$\|U\|$	**Interactions per User**				
		P_1	Q_1	Q_2	Q_3	P_{99}
MovieLens1M	6040	20	44	96	208	909
MovieLens20M	138493	20	35	68	155	1114
Last.FM 2K	1892	10	50	50	50	50
BUBB.Store	6119542	1	1	1	1	12
Casa Cenina	465519	1	1	2	6	335
Just Spotted	340842	1	1	2	4	41
Moodings	562170	1	1	2	5	75

Table 4.3: Item Interaction Characteristics of Experimental Datasets

Dataset Name	$\|I\|$	Interactions per Item				
		P_1	Q_1	Q_2	Q_3	P_{99}
MovieLens1M	3883	0	26	109	330	1783
MovieLens20M	27278	0	3	16	194	14309
Last.FM 2K	17632	1	1	1	3	81
BUBB.Store	51380	0	0	2	32	1906
Casa Cenina	124559	0	4	23	82	837
Just Spotted	4179	0	14	48	143	3063
Moodings	11975	0	29	77	211	2721

nesota. The datasets [43] consist of explicit ratings on rating scale from 1-5 stars, collected through a web interface by the participating users.

Besides the ratings, there are additional metadata about the movies available. For the purposes of our experiments, we picked item attributes $A^I = \{Genres, Title, Year\}$ where dom($Genres$) are tags, dom($Genres$) are unstructured texts, and dom($year$) $= \mathbb{N}$.

4.1.1.2 Last.FM 2K Dataset

This dataset contains social networking, tagging, and music artist listening information from a set of 2K users from Last.fm online music system. It was released in the framework of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011) and at the 5th ACM Conference on Recommender Systems (RecSys 2011) [18].

From the dataset, we picked the artist listening information, containing 92K implicit ratings preprocessed as tuples ($user, art, listening\text{-}count$). We did not extract any item attributes for this dataset.

4.1.1.3 BUBB.Store Dataset

The *BUBB.Store* dataset was provided for our experiments with kind agreement of Lucas Colette, CEO at BUBB WEB STUDIO[1], a Brazilian web agency. BUBB.Store[2] is the flagship product of BUBB WEB STUDIO. The provided dataset contains aggregated users, items, and interactions from multiple web stores. When deployed in production, Recombee Query Language (ReQL) is used to ensure that customers of particular shop will only be recommended items from that shop. For our experiments, we do not use ReQL and allow any item to be recommended to any user, ignoring the fact that there are actually multiple distinct user- and item-sets.

The item attributes are shared across all the products and equal to $A^I = \{StoreID, Brand, Categories, Name\}$, where dom($StoreID$) $= \mathbb{N}$ are the unique shop identifiers,

[1] `https://www.bubb.com.br/`

[2] `https://www.bubbstore.com.br/`

dom(*Brand*) = $\mathbb{N}$ is a set of unique brand identifiers, dom(*Categories*) contains all category strings across all shops, and dom(*Name*) are unstructured texts.

4.1.1.4 Casa Cenina Dataset

Casa Cenina[3] dataset was provided with kind consent of Giovanni Bartoli, CEO at online retailer company specializing at needlework crafts (stitching, quilting, scrapbooking, knitting/crocheting) based in Italy. It is available in four different languages with customers (users) mostly from Italy, France, Spain, Netherlands, and Germany.

The attributes picked for the experiment are $A^I = \{$*Categories*, *Manufacturer*, *Product-Tags*, *Product-Model*$\}$, where dom(*Categories*) = $2^{\mathbb{N}}$ are sets of unique category identifiers (one product may belong to multiple categories), dom(*Manufacturer*) = $\mathbb{N}$ are unique indentifiers of individual manufacturers, dom(*Product-Tags*) = $2^{\mathbb{N}}$ are sets of tags, and dom(*Product-Tags*) = Σ^* are unique model identifiers.

4.1.1.5 Just Spotted and Moodings Datasets

Moodings[4] and *Just Spotted*[5] datasets were provided with kind consent of Nicholas Blicker Larsen, CTO and Co-founder at Design Group ApS, a Denmark based online retailer running the two corresponding E-Commerce websites. The websites specialize at stylish furniture and home accessories, working with brands as Hay, Gubi, &tradition, and Menu. Despite both website focus on different types of products, they share set of similar attributes, from which we picked for the purposes of our experiments the following: $A^I = \{$*Product-name*, *Vendor*, *Type*$\}$, where dom(*Product-name*) are unstructured texts, dom(*Vendor*) = Σ^* are strings uniquely identifying individual vendors, and dom(*Type*) = Σ^* are strings serving as unique category identifiers.

4.1.1.6 Preprocessing

All the aforementioned datasets from Secs. 4.1.1.1, 4.1.1.2, 4.1.1.3, 4.1.1.4, and 4.1.1.5, were preprocessed in unified way. For each dataset, using notation from Sec. 2.1.3, the **interactions** were taken in form of two sets:

- Set of implicit interactions $Y = \{y_1, \ldots, y_k\}$, where $y_j = (u_j, i_j, t_j, d_j)$ such that $u_j \in U$, $i_j \in I$, $t_j \in \mathbb{R}$, $d_j \in \{$*detail-view*, *bookmark*, *cart-addition*, *purchase*, *play*$\}$.
- Set of explicit ratings $X = \{x_1, \ldots, x_\ell\}$, where $x_j = (u_j, i_j, t_j, g_j)$ such that $u_j \in U$, $i_j \in I$, $t_j \in \mathbb{R}$, and $g_j \in [-1, 1]$ is the assigned rating normalized to $[-1, 1]$.

From Y, an implicit rating matrix has been created in compliance with Eq. 2.2 and Eq. 2.3 such that:

[3] `https://www.casacenina.com/`
[4] `https://moodings.com/`
[5] `https://justspotted.dk/`

- Interactions of all types were mixed into the matrix, that is $\varphi(y) = 1 \; \forall y \in Y$.

- Implicit rating matrix $\mathbf{R}^I \in (\mathbb{R} \cup \{?\})^{U \times I}$ was hence simply computed element-wise as $r^I_{u,i} = \Gamma(\{(u_j, i_j, t_j, d_j) \in Y \mid u_j = u \wedge i_j = i\})$ with

$$\Gamma(Y') = \begin{cases} \min\left(\sum_{y' \in Y'} w(y'), 2\right) & \text{if } Y' \neq \emptyset \\ ? & \text{otherwise} \end{cases} \tag{4.1}$$

  using weighting function $w \colon Y \to \mathbb{R}$:

$$w\,(u, i, t, d) = \begin{cases} 0.25 & \text{if } d = \textit{detail-view} \\ 0.5 & \text{if } d = \textit{bookmark} \\ 0.75 & \text{if } d \in \{\textit{cart-addition}, \textit{purchase}, \textit{play}\} \end{cases} \tag{4.2}$$

From set of explicit ratings X, an explicit rating matrix $\mathbf{R}^E \in (\mathbb{R} \cup \{?\})^{U \times I}$ has been created according to Eq. 2.1.

Finally, implicit rating matrix $\mathbf{R}^I$ was merged with explicit rating matrix $\mathbf{R}^E$ into final matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{U \times I}$ using Eq. 2.4.

As a result, for purely explicit rating datasets (MovieLens 1M and MovieLens 20M), the matrix was filled only by explicit rating data, for implicit rating datasets (Last.FM 2K) only by implicit rating data, and for mixed datasets (BUBB.Store, Casa Cenina, Just Spotted, Moodings) by a mixture of both.

4.1.2 Algorithms

For our experiments, we chose the set of learning algorithms corresponding to Sec. 3.2. These algorithms are:

- User-Based k-Nearest Neighbors (UserKnn),

- Item-Based k-Nearest Neighbors (ItemKnn),

- Association Rules (ARs),

- Matrix Factorization (MF), and

- AutoRec.

The implementational details of individual algorithms are discussed below.

4.1.2.1 User-Based k-Nearest Neighbors

We use our own implementation mostly compliant with description in Sec. 2.4.2. The following are some important implementational notes:

1. We use *weighted* version of UserKnn with *non-normalized neighborhood* (Eq. 2.23) and cosine similarity (Eq. 2.24).
2. As a modification to $\mathrm{NN}_k(u)$ in Eq. 2.20, we exclude the users with empty intersection in ratings with the source user $u \in U$. It means that for a given rating matrix $\mathbf{R} \in (\mathbb{R} \cup \{?\})^{U \times I}$, it holds:

$$\forall u, v \in U\colon \{i \in I \mid r_{v,i} \neq ?\} \cap \{i \in I \mid r_{u,i} \neq ?\} = \emptyset \implies v \notin \mathrm{NN}_k(u) \tag{4.3}$$

   This is due to an indexing technique which allows us search for nearest neighbors fast, and it was necessary to use this technique for computationally expensive experiments.
3. In compliance with Eq. 2.20 and the definition of cosine similarity in Eq. 2.24, even the users who rated only a subset of items rated by the source user $u \in U$, are members of $\mathrm{NN}_{\to\infty}(u)$, meaning:

$$\forall u, v \in U\colon \{i \in I \mid r_{v,i} \neq ?\} \subseteq \{i \in I \mid r_{u,i} \neq ?\} \implies v \in \mathrm{NN}_{\to\infty}(u) \tag{4.4}$$

Because the used version is weighted (1) and because of (1), our implementation isn't fully equivalent with the popularity model for $k \to \infty$ as mentioned in Sec. 4.1.2.1, although for high k, the inclination to popular items is still present in limited form.

4.1.2.2 Item-Based k-Nearest Neighbors

Similarly to UserKnn, we also use our own implementation of weighted ItemKnn with similar rules: non-normalized cosine neighborhood (Eq. 2.31) with indexing omitting zero-intersection items from the neighborhood:

$$\forall i, j \in I\colon \{u \in U \mid r_{u,i} \neq ?\} \cap \{u \in U \mid r_{u,j} \neq ?\} = \emptyset \implies i \notin \mathrm{NN}_k(j) \tag{4.5}$$

We experiment with both the interaction similarity (as proposed in [93] and described in Sec. 2.4.3.1) and token similarity (Sec. 2.4.3.2). For the token similarity, we use tokenization fully compliant with Eq. 2.33 and Sec. 2.1.2.

4.1.2.3 Association Rules

We use our own implementation fully compliant with 2.4.4. Because ARs require each item being explicitly relevant or irrelevant, we set a relevance threshold $\theta = 0.25$ to produce binary rating matrix for training.

Because for each dataset, we iterate $s_{\min}$ up to so high granularity that the support corresponds to only single user, it is noteworthy that we had to implement a lot of optimization to make the experiments possible. We maintain the equivalence with the APRIORI

algorithm presented in Alg. 1. However, because for too low $s_{\min}$, the number of rules would easily achieve several billions, we do not discover all the rules in advance. Instead, we search for the rules in lazy manner at the moment of generating recommendations for provided observation set $obs \subseteq I$. Given obs, we prune the search space only for rules $X \Rightarrow Y$ such that $X \subseteq obs$.

4.1.2.4 Matrix Factorization

We use MF for implicit feedback matrices as proposed in [47], with model learning described in Eqs. 2.56, 2.57, and 2.58, using simple *score* function from Eq. 2.46. We used Python `implicit` package[6], which is highly-optimized implementation compliant to [47].

4.1.2.5 AutoRec

We use our own implementation in Python language using the TensorFlow[7] package. Our implementation is fully compliant with Sec. 2.4.6.2. Specifically, we used the U-AutoRec version trained as in 2.74, with *score* function as in Eq. 2.72.

4.2 Capacity Manipulation Experiments

Let us now demonstrate the effect of individual capacity-manipulating hyperparameters on individual learning learning algorithms. For each algorithm, we will present set of measurements of different datasets and discuss the observed behavior.

4.2.1 User-Based k-Nearest Neighbors

Our first set of experiments is done using the UserKnn algorithm. On different datasets, we iterated through different values of $k \in \mathbb{N}$ and $\beta \in [0, 1]$. The results for the industrial datasets are shown in Fig. 4.1, and for publicly available datasets in Fig. 4.2. Instead of separately discussing individual subfigures, most of which are rather similar, let us use few prototypic curves presented in 4.3.

For convenient description, we assigned capital letters to specific segments of the curves. Let us first analyze the curves in Fig. 4.3 (a). We can observe that there are two curves: one without β-biasing (that it $\beta = 0$) and the other with quite high value of $\beta = 1.0$, both curves drawn by iterated k. For the $\beta = 0$ curve, there are three segments: $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$.

[6] `https://github.com/benfred/implicit`

[7] `https://www.tensorflow.org/api_docs/python/`

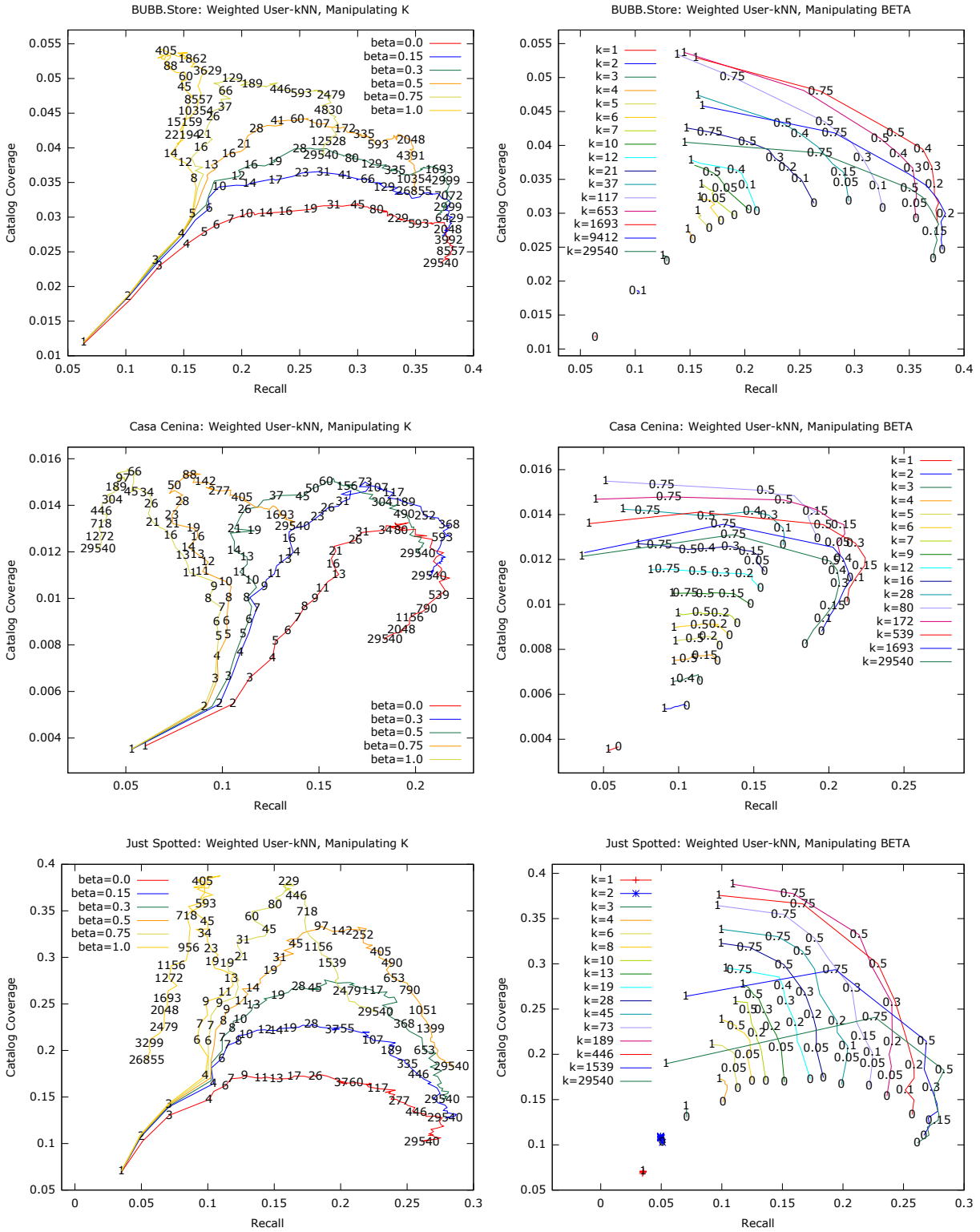

Figure 4.1: UserKnn Experiments on Industrial Datasets.

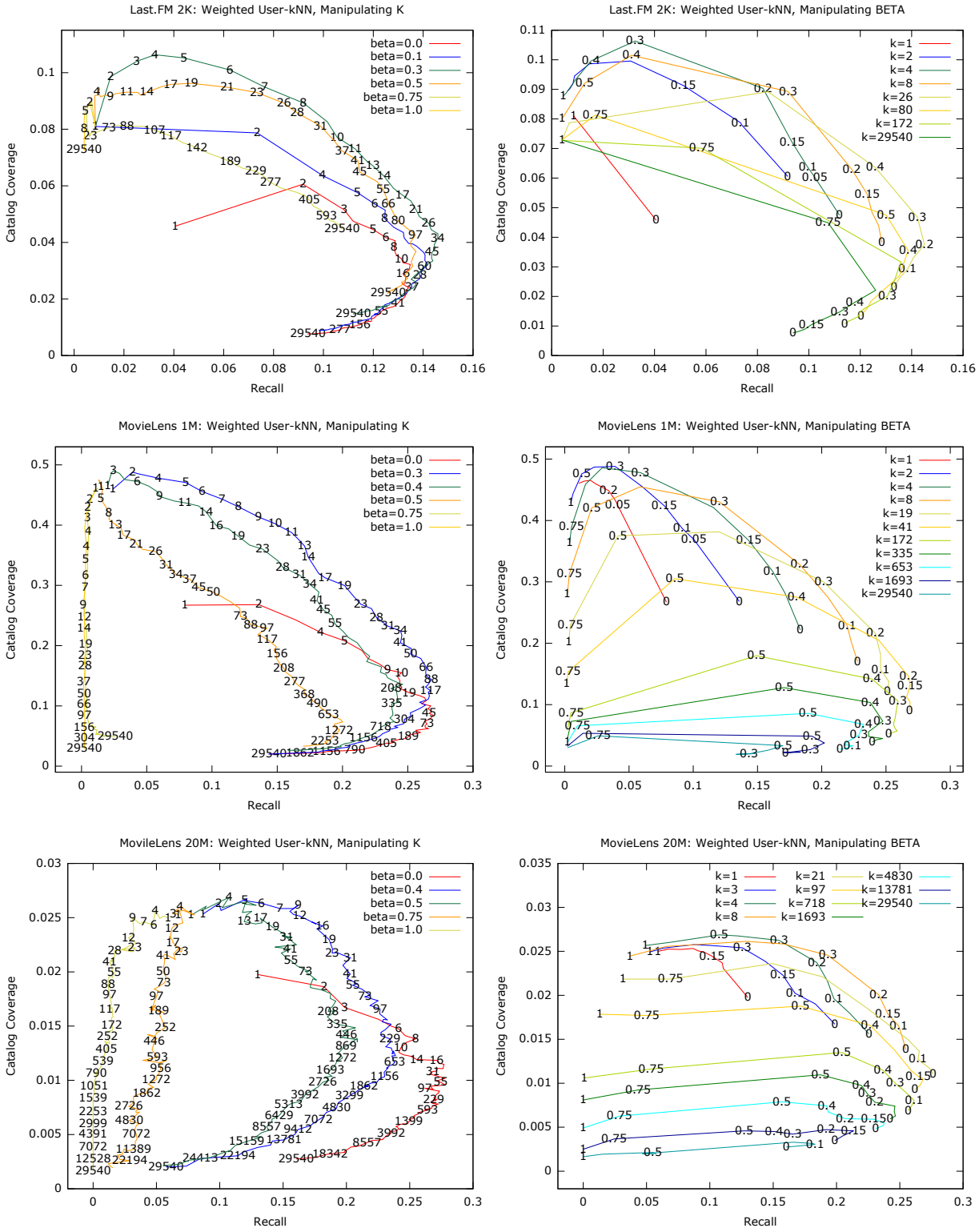

Figure 4.2: UserKnn Experiments on Public Datasets.

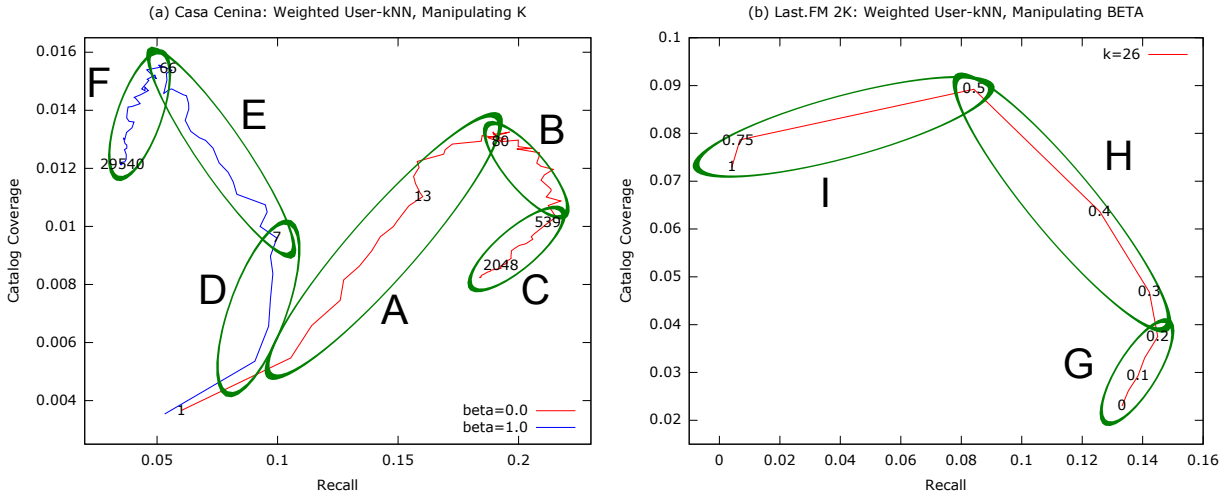

Figure 4.3: Selected Prototypic Curves For The UserKnn Algorithm.

Segment ***A*** starts with $k = 1$ with very low both recall and coverage. As k grows, both measures also grow within ***A***. The reason for this behavior comes from the fact mentioned in Sec. 4.1.2.1: The set $\mathrm{NN}_u(k)$ contains also users whose ratings form a subset of u's ratings. Coming from the definition of cosine similarity, such users are the most similar to u. As a result, the model with too low k fails to recommend the full set of $N = 10$ requested items, simply because the neighboring users don't bring anything new: user u already rated the items that the neighbors have rated. We can say that such low-k model has low user coverage as defined in Eqs. 2.118 and 2.119, which negatively affects both the recall and catalog coverage: Not using all the N available "shots" decreases the chance that there will be a hit for the $\mathit{recall}@N^{\mathrm{UN}}_{\mathrm{LOO}}$, as well as the chance that the catalog coverage set will be extended by something.

With growing k, more and more users with something additional to what u already rated, start appearing in $\mathrm{NN}_u(k)$. In the chosen example curve, there is a saturation at $k \approx 80$. At that point, we can say that the model achieved maximal user coverage for the UserKnn algorithm. Please note the difference between Fig. 4.1 with the industrial datasets and Fig. 4.2 with public datasets. While for industrial datasets, the ***A*** part is always present, for the public datasets, it is very short or missing. This comes from the nature of the data as presented in Table 4.2. The reason is that public datasets has been cleaned not to contain users with too few ratings, and the rating vectors of all the users are very rich. Therefore, it is highly unlikely that one user would form a rating subset of another, and instead, only single nearest neighbor has often all needed $N = 10$ items additional to u's items.

For $k \approx 80$, as the Fig. 4.3 (a) shows, the model is still rather long-tail, prone to over-fitting. It may happen that some noisy items, completely irrelevant to u, are recommended from the nearest neighbors for such a low k. Why the noisy items definitely increase the

catalog coverage, they negatively impact the recall. As we enter the segment ***B*** and keep increasing k, we are decreasing the model capacity, reducing noise, decreasing catalog coverage and increasing recall. This trends keep up to $k \approx 539$ in the figure, where the k start being so high, that the resulting model starts having too low capacity to contain even the important relations in the data.

While with k growing from $k \approx 80$, the catalog coverage keeps constantly decreasing because the set of recommended items keeps reducing to more and more globally popular items, within the ***C*** starting from $k \approx 539$, it is also the recall which starts decreasing. With $k \to \infty$, the model tends to recommend mostly bestsellers. Because we are using weighted version of the algorithm, the popular items from too distant neighbors do not get that high *score*, and together with the implementational detail that users with rating set distinct with u will not appear in $\mathrm{NN}_k(u)$, the model for $k \to \infty$ does not behave as pure popularity model. In the unweighted version with distinct users also appearing in the neighborhood, the curve would get very close to zero coverage, because it would become pure popularity model recommending only very limited set of items.

From the above description, and maybe more conveniently from the figure, it should be obvious that segment ***B*** for the Pareto-optimal front for the given $\beta = 0$ curve. All the models in segments ***B*** and ***C*** are suboptimal, because there exist models with higher catalog coverage and greater or equal recall, or with higher recall with greater or equal catalog coverage.

Let us also describe the other curve in 4.3 (a) for high value of $\beta = 1$, consisting on segments ***D***, ***E***, and ***F***. This curve behaves differently, but it is the very high value of β which causes that candidate items $i \in I$ obtaining $score(u, i) > 0$, get boosted by their inverse popularity, and hence mostly the least popular of them appear in the resulting Top-N recommendation set.

The growth of segment ***D*** starting from very low k, has similar justification as the segment ***A*** in the $\beta = 0$ curve. The model starts recommending something rather than nothing, which logically leads to improvement in both the recall and the coverage. However, because of the very high β-boosting, the models tends to pick "the worst" (meaning the most random, globally unknown) items, and hence as the set of candidate $score(u, i) > 0$ items extends, the recall starts getting lower. This happens at $k \approx 7$ in the figure, where the transition to segment ***E*** begins.

Within ***E***, the trend of "picking the worst candidates by high β" keeps intensifying, as more and more neighbors generate more candidate $score(u, i) > 0$ items, and from these, more marginal, random, noisy items can be picked. Therefore, the catalog coverage keeps increasing withing ***E***, but the recall deteriorates. Another break points is at $k \approx 66$ in the figure. While the curves are slightly affected by noise, it should be obvious that this is basically very similar k where the user coverage got saturated also it the $\beta = 0$ curve. Entering segment ***F***, the catalog coverage starts to decrease. This is caused by the fact that the more candidate items, the more noisy items to be "picked" by high $\beta = 1$, but the set of marginal, noisy items in the dataset is limited. With growing k, it happens that these most marginal items start appearing in neigborhoods for more and more users, and being constantly picked. Hence the decreasing coverage: the model becomes "inverse

bestseller", more and more focusing on a limited set of marginal items. This is somehow symmetric to segment $\boldsymbol{C}$ with popularity replaced by "anti-popularity" due to $\beta = 1$.

The two curves on Fig. 4.3 (a) represent two extremes with $\beta = 1$ being obviously a bad choice. But looking at results for individual datasets on Figs. 4.1 and 4.2, it is clear that for moderately high β, the original model gets improved toward new Pareto-optimal points. For smaller values of β, the segment $\boldsymbol{F}$ is not present and it rather turns to $\boldsymbol{B}$. Let us explain this using Fig. 4.3 (b). On a different dataset, for fixed k, we present a prototypic recall-coverage curve when iterating over increasing β.

We can see, that the chosen dataset, the model with $k = 26$ has rather low capacity. It seems like weighted cosine similarity, used in the *score* function, is unable to pick enough long-tail items even from small neighborhoods. In segment $\boldsymbol{G}$, with $\beta = 0$, we obtain the original model. With growing β, the capacity of the original high-bias, low-variance model, is increased, which improves both the recall and the coverage. Note that some curves in Figs. 4.1 and 4.2 are missing the $\boldsymbol{G}$ segment, simply because the original, non-β-boosted models has high capacity already. Segment $\boldsymbol{G}$ can be seen as equivalent to segment $\boldsymbol{C}$ when iterating over k.

Reaching $\beta \approx 0.2$ in Fig. 4.3 (b) and approaching segment $\boldsymbol{H}$, the β-biasing starts being quite high, and while the catalog coverage keeps growing with k, the recall deteriorates as the increasing capacity shifts the model more and more towards the long-tail and the recommending items are become too marginal. The decrease of recall comes from the (apparently correct) MNAR assumption, that impopular items are unlikely to be present in the $r^+(u)$ of the given testing user. Segment $\boldsymbol{H}$ can be seen as equivalent to segment $\boldsymbol{B}$ when iterating over k.

Finally, at $\beta \approx 0.5$, the boosting of marginal items is becoming so high, that we are reaching the "antipopularity" model, focusing on a limited set of marginal items, and catalog coverage hence starts decreasing as well.

Besides the UserKnn results presented in Figs. 4.1 and 4.2, let us also present additional interesting results from a bachelor thesis [69] supervised by the author of this thesis. In [69], optimization of the UserKnn algorithm through the use of technique known as Locality-Sensitive Hashing (LSH) [48] is proposed to make the construction $\mathrm{NN}_k(u)$ asymptotically bound to sublinear time with $|U|$, and studied in relation to recall-coverage curves.

The idea of using LSH with cosine similarity (Eq. 2.24) in UserKnn lies in generating a random projection matrix $\mathbf{A} \in \mathbb{R}^{f \times I}$ (for relatively small $f \approx 10$), typically drawn from normal distribution $\mathcal{MN}_{f \times U}(\mathbf{0}, \mathbf{I}, \mathbf{I})$. From the rating matrix $\mathbf{R} \in \mathbb{R}^{U \times I}$ with $? \to 0$ replacements, matrix $\mathbf{X} \in \mathbb{R}^{f \times U}$ is computed as $\mathbf{X} = \mathbf{A} \cdot \mathbf{R}^T$. Values of $\mathbf{X}$ are subsequently binarized to matrix $\mathbf{H} \in \{0, 1\}^{f \times U}$ such that

$$\forall u \in U, \ell \in \{0, \ldots, f\} \colon h_{u,\ell} = \begin{cases} 1 & \text{iff } x_{u,\ell} \geq 0 \\ 0 & \text{iff } x_{u,\ell} < 0 \end{cases} \tag{4.6}$$

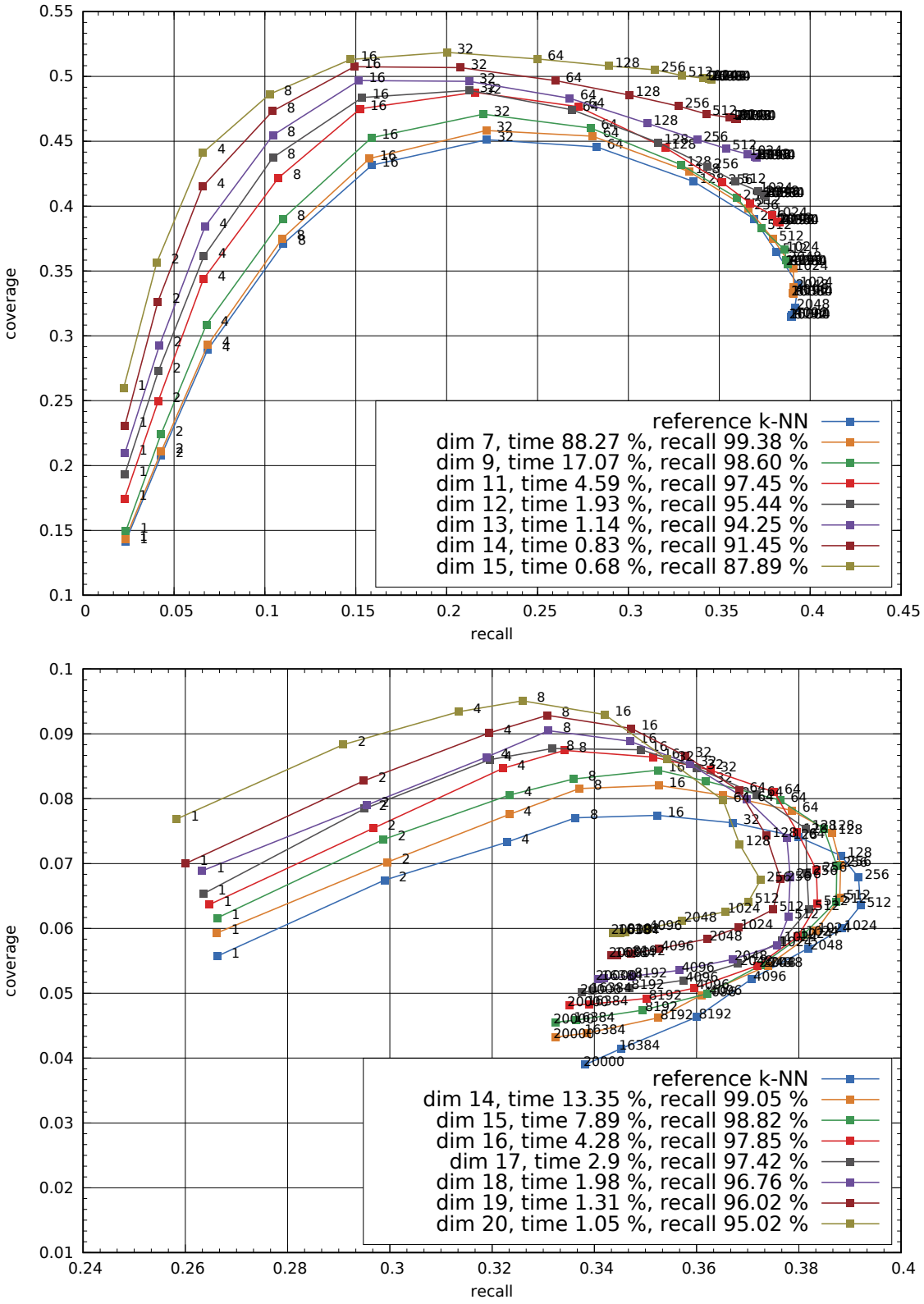

Figure 4.4: UserKnn Results Using Locality-Sensitive Hashing [69].

Row vector $\mathbf{h}_{*,u}$ for user u in the resulting matrix represents a hash of the bucket that u belongs to. In total, there are 2^f buckets, and when searching for $\mathrm{NN}_k(u)$, only the users from the corresponding $\mathbf{h}_{*,u}$ bucket (and possibly also the buckets closest to it in the Hamming distance) are considered as candidates. Fig. 4.4 shows the results achieved on the chosen datasets. The shape of curves on approximate neighborhood is preserved, typically leading to slightly reduced recall, but also improved Pareto-optimal front when considering also the coverage.

4.2.2 Item-Based k-Nearest Neighbors

Another set of experiments was performed with the ItemKnn algorithm. Similarly to UserKnn, we iterated the learning algorithm over different values of $k \in \mathbb{N}$ and $\beta \in \mathbb{R}$. Besides using the rating cosine similarity (Sec. 2.4.3.1), we also investigated the behavior for the token similarity.

4.2.2.1 Rating (Interaction) Similarity

The results for ItemKnn with rating similarity are shown in Fig. 4.6, To thoroughly explore the space, we iterated different values of $\beta \in [-10, 10]$, showing the behavior also for high biases. Let us again discuss some prototypic curves. These are shown in Fig. 4.5.

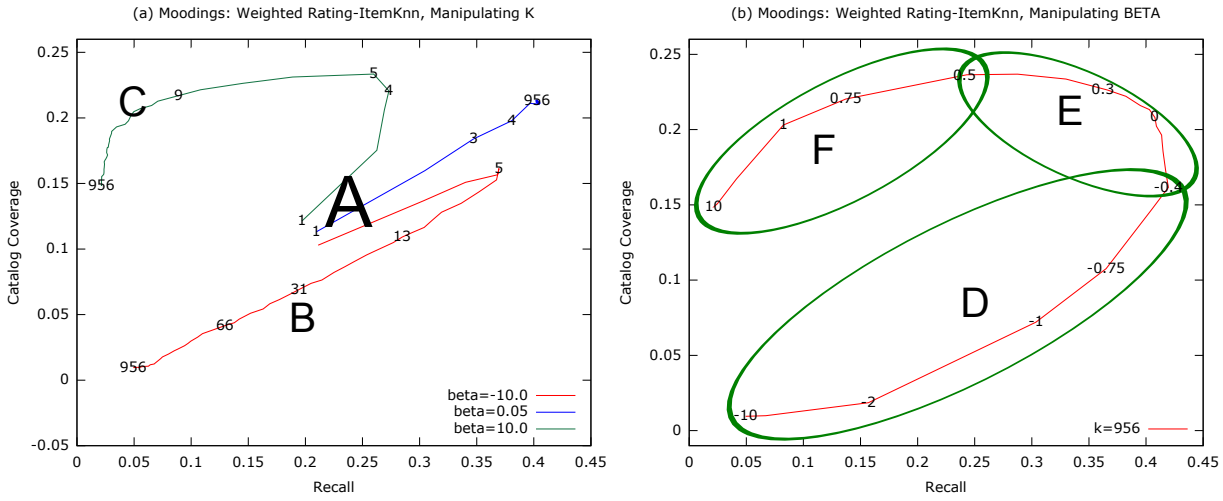

Figure 4.5: Selected Prototypic Curves For The ItemKnn Algorithm.

Looking at Fig. 4.5 (a), similarly to UserKnn, there is segment ***A*** starting from $k = 1$, where both the recall and the coverage grow with k. In the figure, there are 3 different curves for different values of β. All the three curves share the ***A*** segment. With more items in the neighborhoods of individual source user's rated items, the higher chance that the model will be able to generate the full set of $N = 10$ items in the Top-N recommendation.

Again, more recommended items increase the chance of a hit within the leave-one-out recall, same as adding something to the catalog coverage set build over the group of validation users.

Depending on the value of β, segment $\boldsymbol{A}$ may either form the whole curve (for β close to 0), or may continue to segments $\boldsymbol{B}$ or $\boldsymbol{C}$. For too low value of β (segment $\boldsymbol{B}$), with growing k, there is again a transition towards the popularity model, which results in nearly zero coverage, as well as very low recall. For too high value of β the transition is into the "anti-popularity" model, for the same reasons as in the UserKnn model.

Consider the curve iterated over β in Fig. 4.5 (a), it is noteworthy that unlike with UserKnn, the curve tends to start at a point of balance between the recall and the coverage. This is because ItemKnn is rather long-tail recommender by default. This is because the algorithm is by design based on recommending very similar item to these in the user's observation. Therefore, in compliance with the presented results, the $\beta = 0$ point often lies withing the $\boldsymbol{E}$ segment, which is a Pareto-optimal front for the whole β-iterated curve. Putting $\beta < 0$, we start moving toward low capacity model with higher recall and lower coverage, and vice versa for $\beta > 0$.

The interpretation to the $\boldsymbol{D}$ and $\boldsymbol{F}$ segments is the same as for UserKnn segments $\boldsymbol{G}$ and $\boldsymbol{I}$, respectively, in 4.3 (b).

4.2.2.2 Token Jaccard Similarity

For the attribute-based ItemKnn, we used the token similarity (Sec. 2.4.3.2) on top item attributes summarized in Table 4.1, described for individual datasets in Secs. 4.1.1.1, 4.1.1.3, 4.1.1.4, and 4.1.1.5, and tokenized through a process described in Sec. 2.1.2.

The experiment results are shown in Fig. 4.7. As seen on the examples of Just Spotted and Moodings, the token-based ItemKnn can hardly compete with the rating-based one, but the curves share very similar behavior and shape. This is because the fact, that the algorithm tends to recommend items very similar to the observed ones, does not hold only for interaction similarity, but logically, for the token similarity as well. Looking for items with very similar attributes leads to long-tail recommendation by default. Again, using β, the capacity can be manipulated, shifting to more long-tail or to more popular items being recommended.

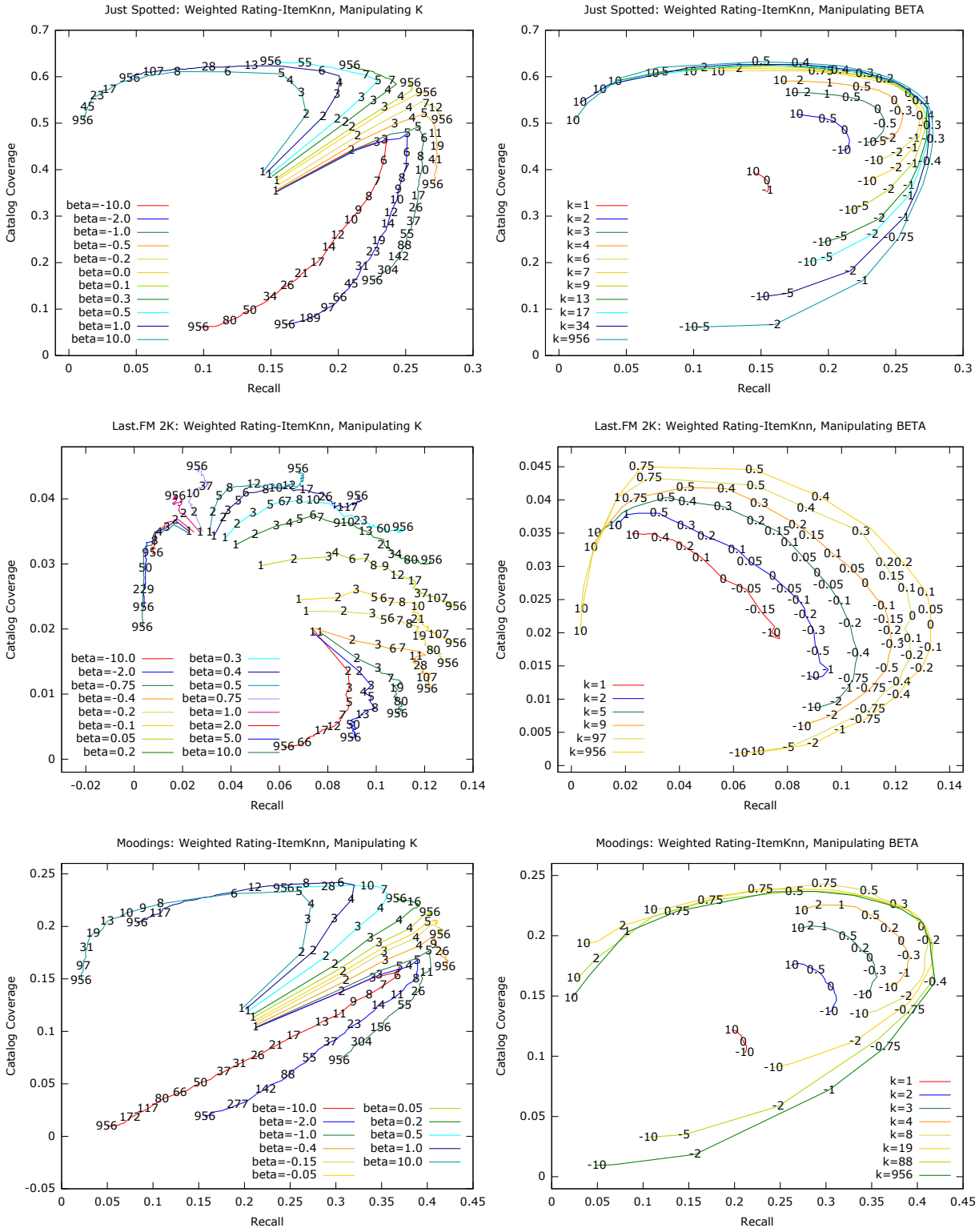

Figure 4.6: ItemKnn Experiments Using Rating Cosine Similarity

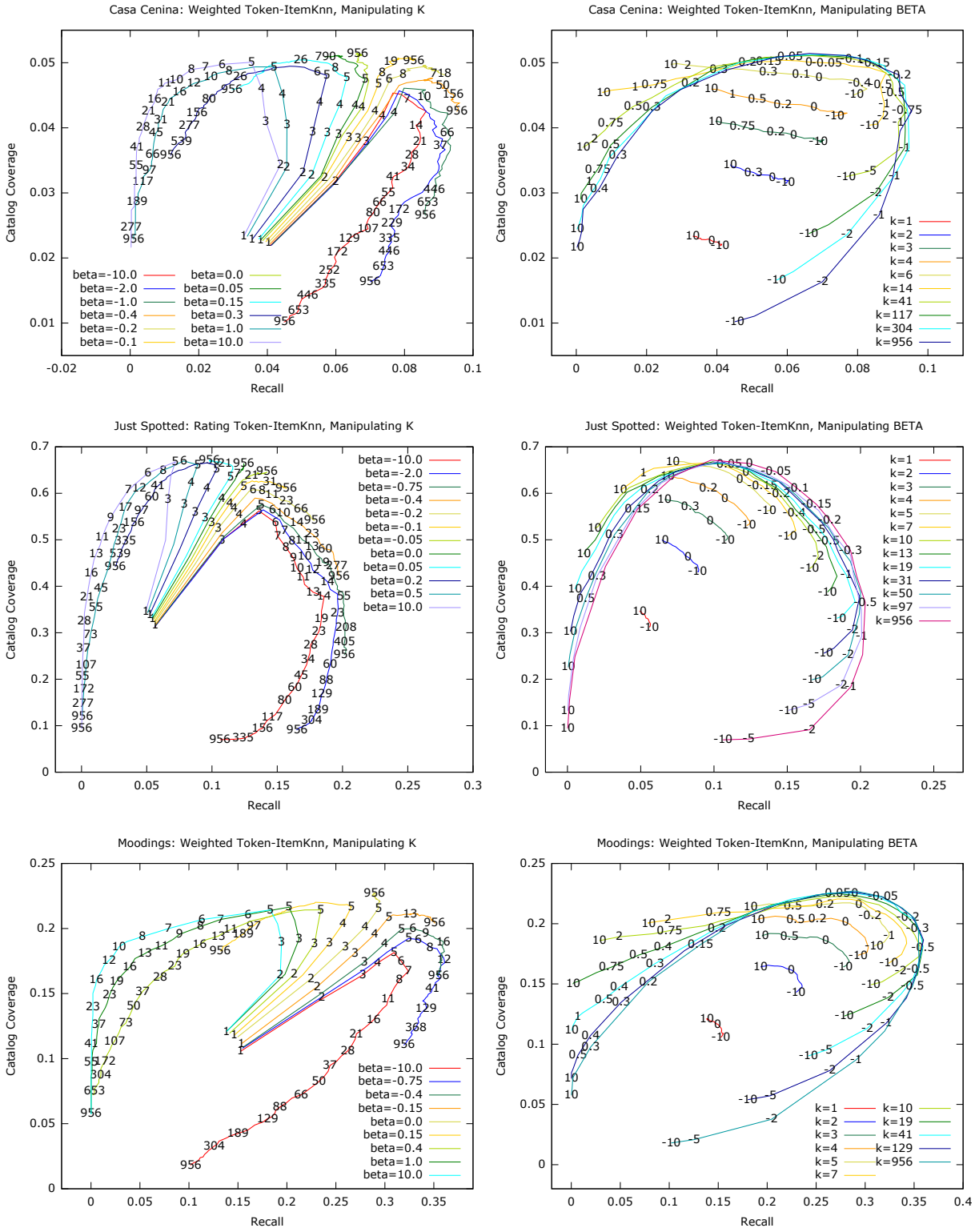

Figure 4.7: ItemKnn Experiments Using Token Jaccard Similarity

4.2.3 Association Rules

Our experiments continue with Association Rules, exploring both the best-rule and weighted voting method. The results for the best-rule method are shown in Fig. 4.9, and for weighted voting method in Fig. 4.10. As the rule quality measure, we used the confidence (Eq. 2.41) in compliance with [91], but as shown in Sec. 3.2.4, using $\beta \in [0, 1]$ can be seen as a transition to lift (Eq. 2.42).

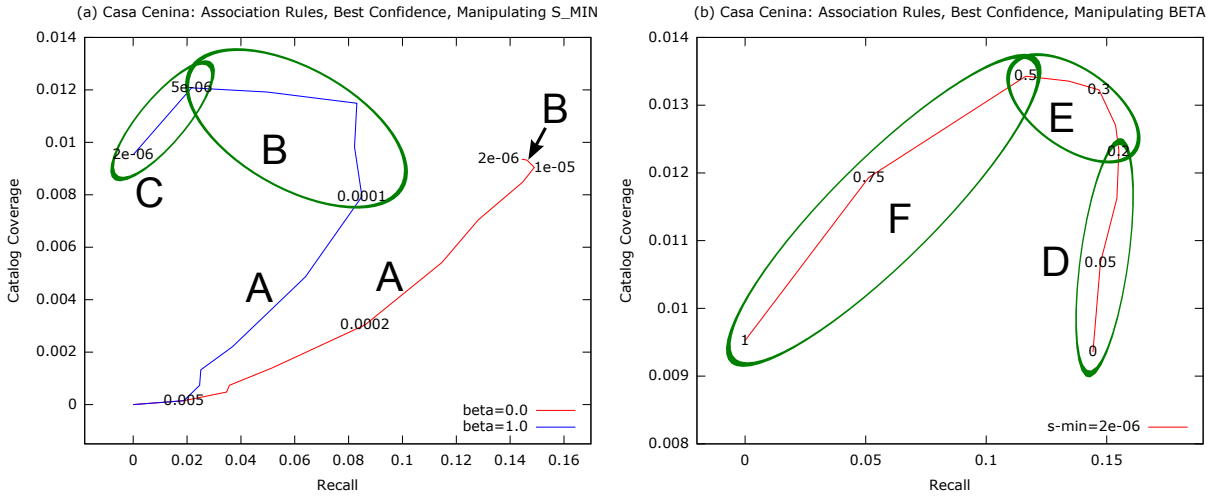

Figure 4.8: Selected Prototypic Curves For The Association Rules.

The segments shown in 4.8 are very similar to segments commented in the neighborhood-based algorithms. High value of $s_{\min}$ can be thought as equivalent to high value of k in the UserKnn algorithm. As discussed in 3.2.4, high $s_{\min}$ causes that only the most important inter-item relations are learned, while low $s_{\min}$ leads to high granularity of discovered rules, and therefore to high-capacity, long-tail model. Segments ***A*** and ***B*** in 4.8 (a) are equivalent to segments ***C*** and ***B***, respectively, in 4.3 (a) for UserKnn. The β-iterated curve in 4.8 (b) does not require any additional comments either, since it behaves in the same way as for neighborhood-based algorithms. The only difference is in the exact shape, which is caused by different underlying model, and of course depends on given dataset.

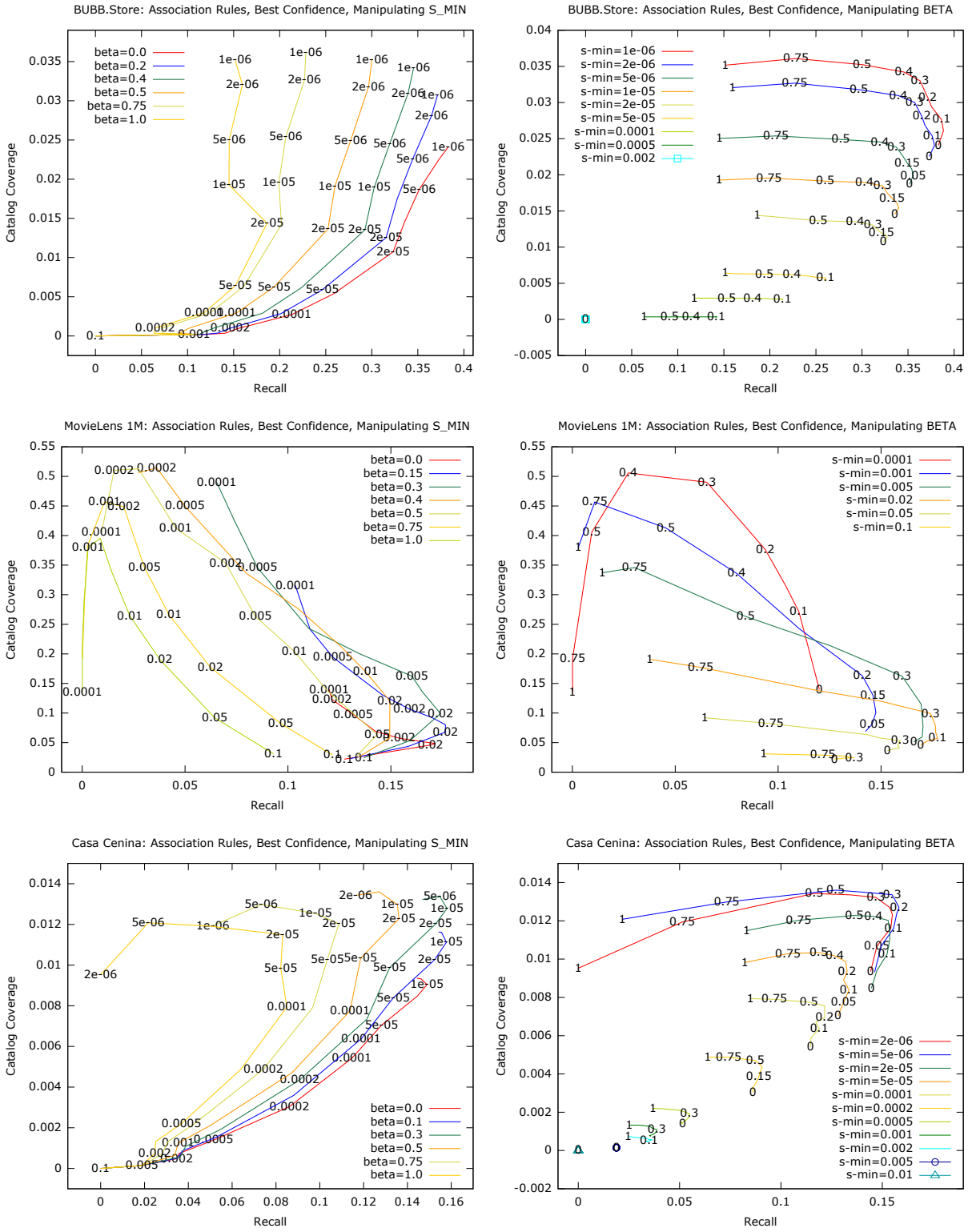

Figure 4.9: Association Rules Experiments Using Best-Confidence Method

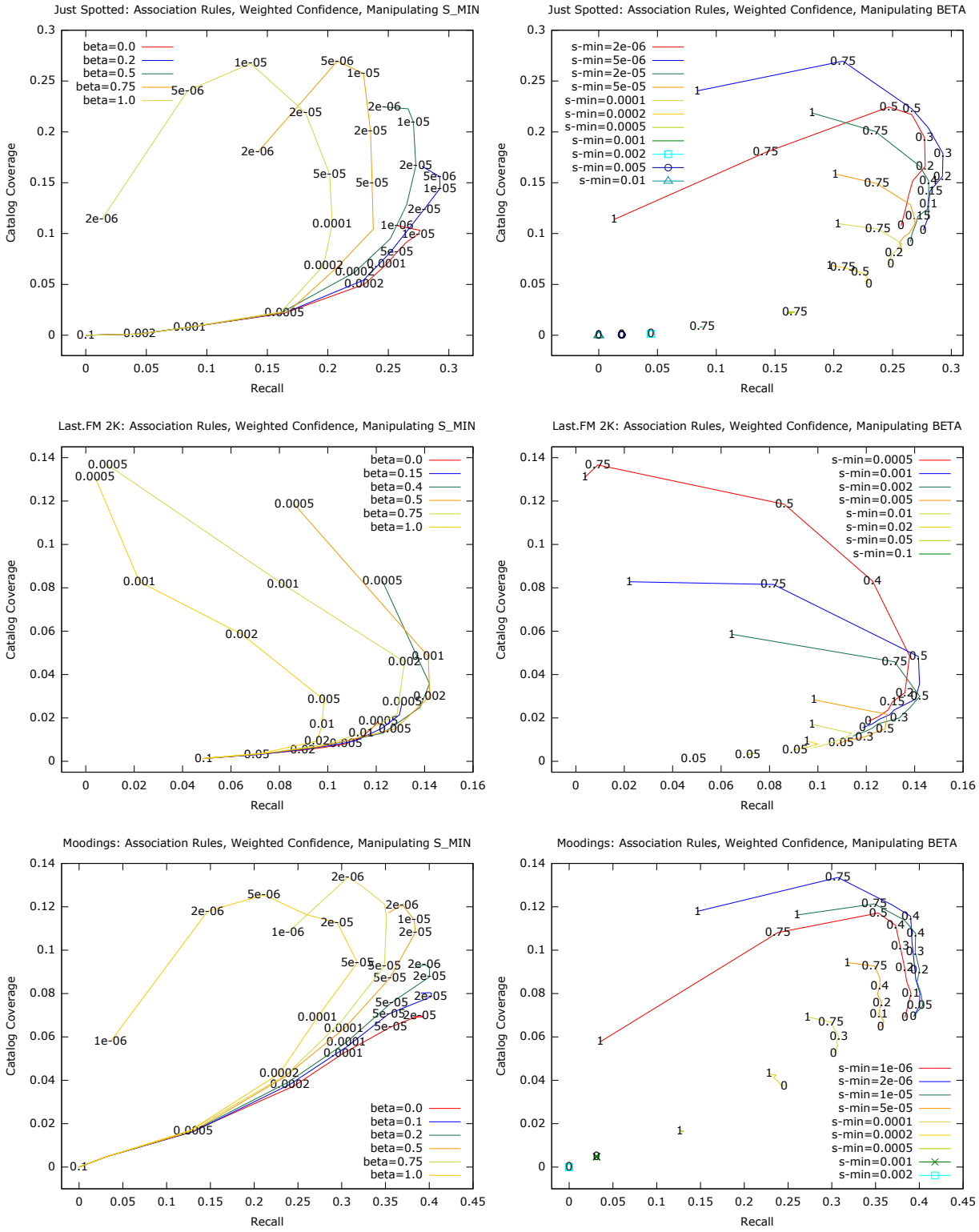

Figure 4.10: Association Rules Experiments Using Weighted-Confidence Voting Method

4.2.4 Matrix Factorization

We have also done a number of experiments with the Matrix Factorization. Fig. 4.12 shows the results for the Moodings and the Last.FM 2K datasets, iterated over different capacity-manipulating hyperparameters: the number of factors $f \in \mathbb{N}$, the regularization $\lambda \in \mathbb{R}$, and, as usually, $\beta \in \mathbb{R}$. Unlike for previous algorithms, the capacity hyperparameterization space has 3 dimensions, making the search for interesting ares slightly more difficult.

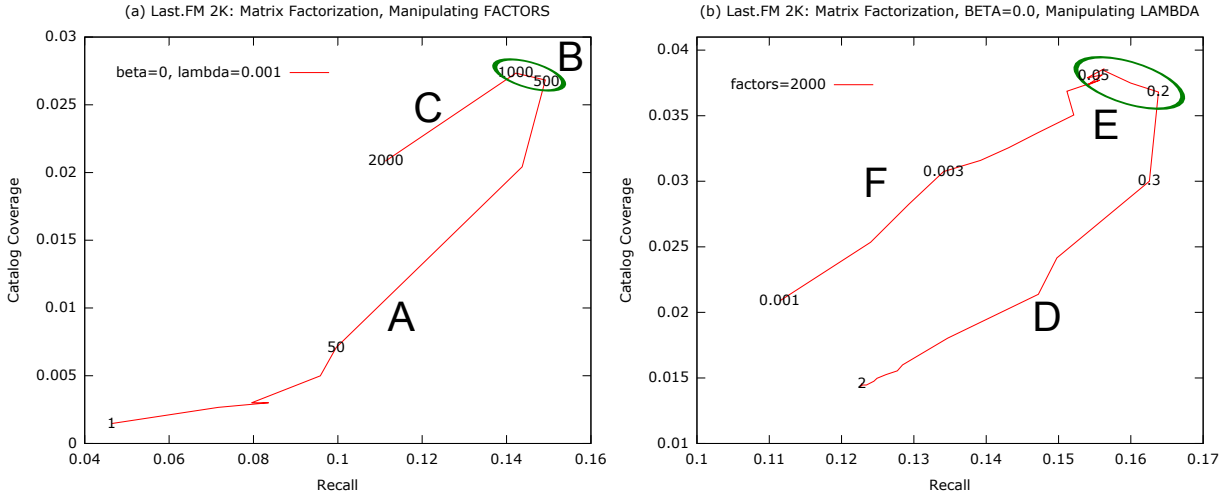

Figure 4.11: Selected Prototypic Curves For The Matrix Factorization.

Because the behavior for β is algorithm-independent as shown above, the prototypic curves for MF in Fig. 4.11 include only the number of factors f and regularization λ. Segment ***A*** for the number of factors in 4.11 (a) starts at low capacity, popularity-based model for $f = 1$, and traverses towards higher-capacity models with in increasing f, growing in both the recall and the coverage. This is equivalent for transition from $s_{\min} = 1$ in case of ARs, and from $k \to \infty$ in case of UserKnn model. The Pareto-optimal front is segment ***B***.

What deserves separate comment is ***C*** segment, which decreases in coverage despite $\beta = 0$. In this case, it is the low chosen value of λ which causes this overfit. Given too high number of factors without enough regularization, the model overfits to the provided rating matrix through too high absolute values of individual elements in matrices $\mathbf{P} \in \mathbb{R}^{f \times U}$ and $\mathbf{Q} \in \mathbb{R}^{f \times I}$. If for some items the values go too high (lowering error on the training set), the result on the testing set will be that dot product with latent vectors with such items will be the highest for majority user latent vectors build for users from the testing set. This impact of λ is shown in 4.11 (b), where segment ***D*** represents underfitting (popular items), segment ***E*** the desired Pareto-optimal front, and segment ***F*** overfitting (noisy items).

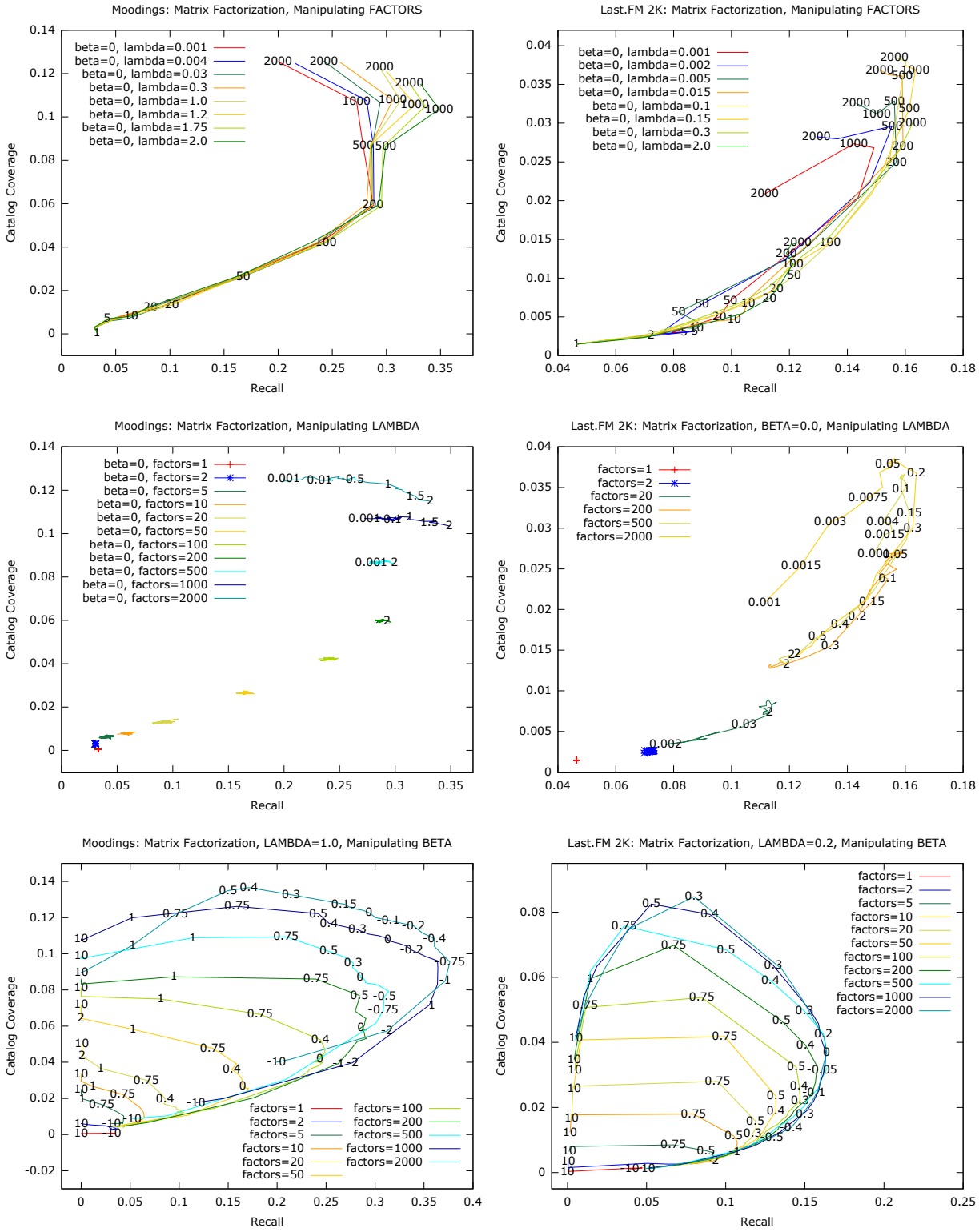

Figure 4.12: Matrix Factorization Experiments

4.2.5 AutoRec

Finally, we present results for the AutoRec algorithm on the MovieLens 1M dataset in Figs. 4.13 and 4.14. Since the algorithm is basically a generalized matrix factorization, the behavior for individual parameters is also similar. Small number of neurons in the bottleneck leads to popularity-based model, raising their number increases model capacity. We iterated only up to 200 neurons, so there wasn't overfit to noisy achieved, but it's expected that for higher number of neurons, the coverage would start decreasing. Using λ regularization, we show all the three well-known segments: Underfitting for too high λ, Pareto-optimal front for decreasing values of λ, and overfitting to noisy items for very low $\lambda = 10^{-7}$. For β, the behavior is the same as for other algorithms.

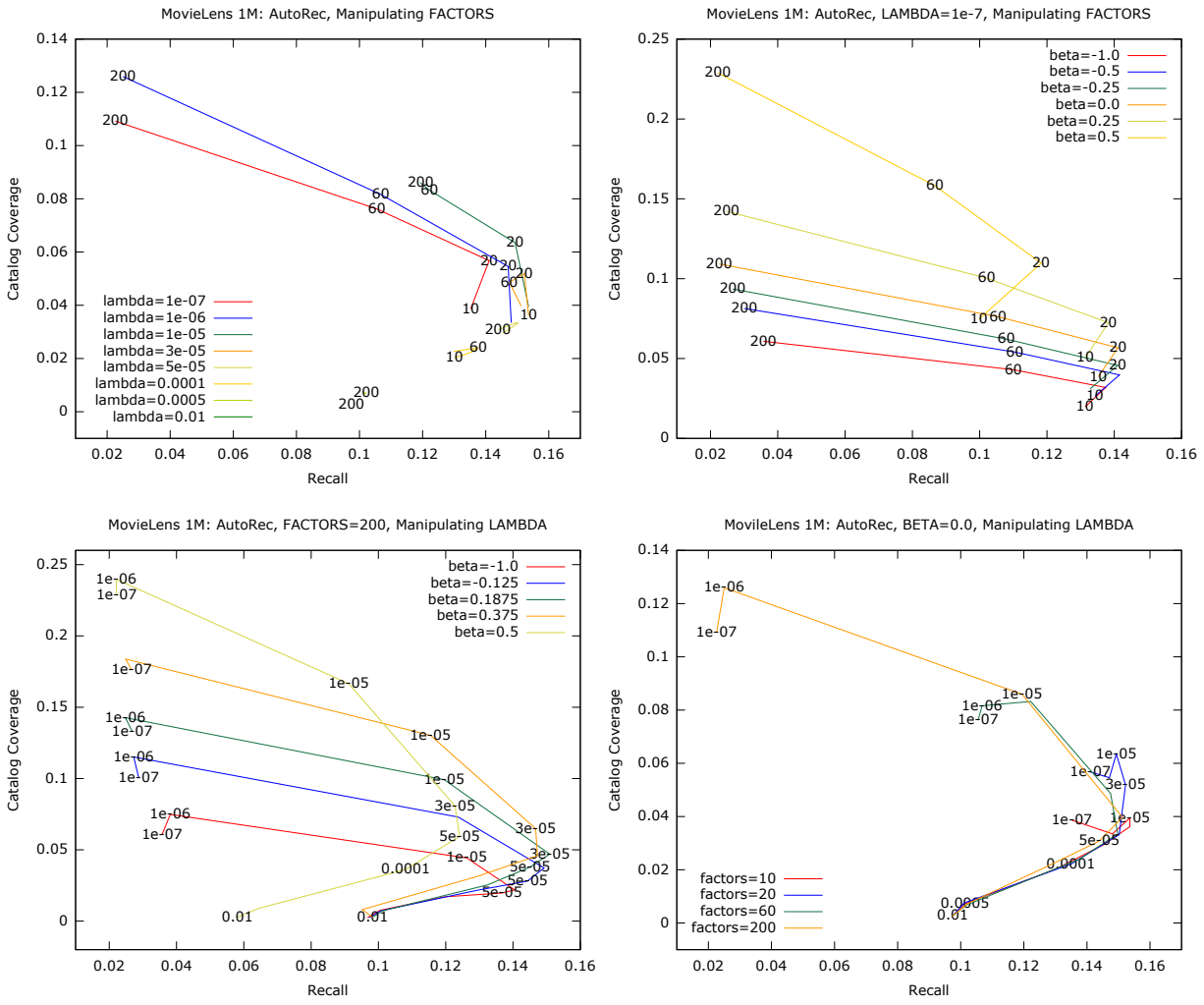

Figure 4.13: AutoRec Experiments on MovieLens 1M Dataset for f and λ

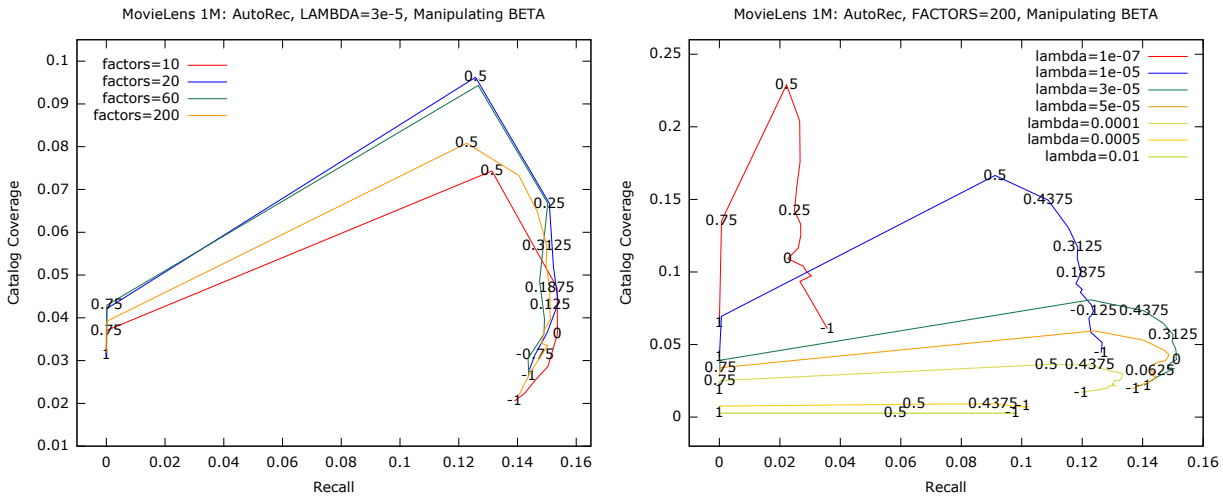

Figure 4.14: AutoRec Experiments on MovieLens 1M Dataset for β

Chapter 5

Conclusions

In this final chapter, we will conclude the whole thesis, including its main contributions and opportunities for future research.

5.1 Summary

In this thesis, we addressed the long-tail recommendation problem from both the algorithmic and evaluation perspective. Most importantly, we proposed controlling the magnitude of long-tail recommendations generated by models through the manipulation with capacity hyperparameters of learning algorithms. For multiple state-of-the-art algorithms, we defined such hyperparameters. Thanks to summarizing all the algorithms under the common framework of the *score* function, we also generalized popularity-based regularization proposed in [101] for Matrix Factorization, to all the presented models. We proposed searching for Pareto-optimal states in the Recall-Coverage plane as the right way to search for long-tail, high-accuracy models. On the set of exhaustive experiments, we empirically demonstrated the corectness of our theory on a mixture of public and industrial datasets for 5 different algorithms and their different versions.

5.2 Contributions of the Dissertation Thesis

- Unified framework for generating rating matrices from mixed data sources, including both the explicit and implicit ratings.
- Summarizing multiple state-of-the-art recommendation algorithms under the common framework based on the $score: I \times U \to \mathbb{R}$ function.
- Generalizing Collaborative Filtering ItemKnn algorithm as proposed in [93] to $sim: I^2 \to \mathbb{R}$ functions other than rating cosine similarity. The newly proposed similarity functions include the *tokenized attributes* similarity, *embedding* similarity,

and *predicted* similarity. This allows using the algorithm for Content-Based recommendation as well.

- Association Rules for recommendation. While this is type of model didn't receive enough attention in the research community, we proposed unified framework for rule-based recommendation, including the best-rule method, novel method of weighted voting, and different rule-quality measures. We also showed how the β-boosting (popularity regularization) applied to ARs translates to continous transition from *confidence* to *lift*. In the experiments, we showed that rule-based recommenders can compete to other state-of-the-art models.

- Survey of Matrix Factorization methods and Deep Learning approaches in Recommender Systems.

- Formal framework for hyperparameterizable learning algorithms producing recommendation models. Generalized definion of validation reward function, capturing not only accuracy measures, but other measures (inluding catalog and user coverage) as well. Unifying UserKnn, ItemKnn, ARs, MF, and AutoRec under this framework.

- Survey to Top-N evaluation in Recommender Systems.

- Generalization of popularity-based regularization, proposed in [101] for one specific MF algorithm, to all the algorithms presented in the State-of-the-Art chapter.

- Proposing model capacity manipulation as a method of controlling magnitude of long-tail recommendations.

- Defining model capacity manipulation parameters for UserKnn, ItemKnn, ARs, MF, and AutoRec.

- Proposing multi-objective top-N evaluation as searching for Pareto-optimal front in the hyperparameter space of learning algorithms.

- Proposing recall-coverage plane as highly relevant for optimizing long-tail recommendations.

- Exhaustive set of experiments on a mixture of 7 public and industrial datasets. Thorough investigation of manipulation with algorithm-dependent capacity parameters and the response in the recall-coverage plane.

- For UserKnn, showing that recall-coverage behavior is preserved even for Locality-Sensitive Hashing (LSH).

5.3 Future Work

The author of the dissertation thesis suggests to explore the following:

- It would be interesting to include additional reward functions other than recall and coverage, such as intra-recommendation diversity, novelty, and serendipity.
- Set of online A/B testing experiments should be performed to further investigate the correlation between recall, coverage, and measures like CTR and CR. We performed few such experiments already with results suggesting that models with lower recall to the benefit of higher coverage lead to better online results.

In the Appendix A, we include our recent paper on comparison between online and offline results in RS. It would be particularly interesting to combine the two methods together.

Bibliography

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering*, (6):734–749, 2005.

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[4] R. Agrawal and R. Srikant. Mining sequential patterns. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, pages 3–14. IEEE, 1995.

[5] Y. Aytar, C. Vondrick, and A. Torralba. Soundnet: Learning sound representations from unlabeled video. In *Advances in Neural Information Processing Systems*, pages 892–900, 2016.

[6] P. J. Azevedo and A. M. Jorge. Comparing rule measures for predictive association rules. In *Proceedings of the 18th European Conference on Machine Learning*, ECML '07, pages 510–517, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] M. Bajer. Framework pro vyhodnocování úspešnosti algoritmu castých sekvencí v doporucovacích systémech., 2017.

[8] C. Basu, H. Hirsh, W. Cohen, et al. Recommendation as classification: Using social and content-based information in recommendation. In *Aaai/iaai*, pages 714–720, 1998.

[9] J. Beel, M. Genzmehr, S. Langer, A. Nürnberger, and B. Gipp. 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*, pages 7–14. ACM, 2013.

[10] R. M. Bell and Y. Koren. Lessons from the netflix prize challenge. *SIGKDD Explor. Newsl.*, 9(2):75–79, Dec. 2007.

[11] R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 43–52, Oct 2007.

[12] J. Bennett, S. Lanning, and N. Netflix. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[13] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *Icml*, volume 98, pages 46–54, 1998.

[14] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003.

[15] T. Bogers and A. Van den Bosch. Recommending scientific articles using citeulike. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 287–290. ACM, 2008.

[16] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, Nov. 2002.

[17] E. Campochiaro, R. Casatta, P. Cremonesi, and R. Turrin. Do metrics make recommender algorithms? In *Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on*, pages 648–653. IEEE, 2009.

[18] I. Cantador, P. Brusilovsky, and T. Kuflik. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, New York, NY, USA, 2011. ACM.

[19] P. Castells, N. J. Hurley, and S. Vargas. Novelty and diversity in recommender systems. In *Recommender Systems Handbook*, pages 881–918. Springer, 2015.

[20] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 7–10, New York, NY, USA, 2016. ACM.

[21] Y.-H. Chien and E. I. George. A bayesian model for collaborative filtering. In *AISTATS*, 1999.

[22] M. Claesen and B. D. Moor. Hyperparameter search in machine learning. *CoRR*, abs/1502.02127, 2015.

[23] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM.

[24] W. B. Croft, D. Metzler, and T. Strohman. *Search engines: Information retrieval in practice*, volume 283. Addison-Wesley Reading, 2010.

[25] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296. ACM, 2010.

[26] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

[27] E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-time top-n recommendation in social streams. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 59–66. ACM, 2012.

[28] X. Dong, L. Yu, Z. Wu, Y. Sun, L. Yuan, and F. Zhang. A hybrid collaborative filtering model with deep structure for recommender systems, 2017.

[29] P. Forbes and M. Zhu. Content-boosted matrix factorization for recommender systems: experiments with recipe recommendation. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 261–264. ACM, 2011.

[30] S. Funk. Netflix update: Try this at home. `http://sifter.org/~simon/journal/20061211.html`, 2006.

[31] C. Gallicchio, A. Micheli, and L. Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87–99, 2017.

[32] F. Garcin, B. Faltings, O. Donatsch, A. Alazzawi, C. Bruttin, and A. Huber. Offline and online evaluation of news recommender systems at swissinfo. ch. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 169–176. ACM, 2014.

[33] M. Ge, C. Delgado-Battenfeld, and D. Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 257–260. ACM, 2010.

[34] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 69–77, New York, NY, USA, 2011. ACM.

[35] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[36] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudik, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[37] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, Dec. 1992.

[38] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *information retrieval*, 4(2):133–151, 2001.

[39] C. A. Gomez-Uribe and N. Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):13, 2016.

[40] M. Hak. Porovnání různých metod měření úspěšnosti v kolaborativním filtrování, 2014.

[41] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, May 2000.

[42] A. S. Harpale and Y. Yang. Personalized active learning for collaborative filtering. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 91–98. ACM, 2008.

[43] F. M. Harper and J. A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, Dec. 2015.

[44] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[45] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[46] T. Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, Jan. 2004.

[47] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, Dec 2008.

[48] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[49] C. Johnson. Algorithmic music discovery at spotify. `https://www.slideshare.net/MrChrisJohnson/algorithmic-music-recommendations-at-spotify`, 2014.

[50] A. Karatzoglou, L. Baltrunas, and Y. Shi. Learning to rank for recommender systems. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 493–494. ACM, 2013.

[51] G. Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 247–254. ACM, 2001.

[52] P. Kasalický. Content-based recommendation model trained using interaction similarity, 2018.

[53] N. Koenigstein and U. Paquet. Xbox movies recommendations: Variational bayes matrix factorization with embedded feature selection. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 129–136, New York, NY, USA, 2013. ACM.

[54] I. Kononenko. Combining decisions of multiple rules. *Artificial Intelligence V: Methodology, Systems, Applications*, pages 87–96, 1992.

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

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

[57] J. Langford, A. J. Smola, and M. Zinkevich. Slow learners are fast. In *Proceedings of the 22Nd International Conference on Neural Information Processing Systems*, NIPS'09, pages 2331–2339, USA, 2009. Curran Associates Inc.

[58] Y. LeCun, Y. Bengio, and G. E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[59] C. W.-k. Leung, S. C.-f. Chan, and F.-l. Chung. Applying cross-level association rule mining to cold-start recommendations. In *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops*, WI-IATW '07, pages 133–136, Washington, DC, USA, 2007. IEEE Computer Society.

[60] Q. Li and X. Zheng. Deep collaborative autoencoder for recommender systems: A unified framework for explicit and implicit feedback. *CoRR*, abs/1712.09043, 2017.

[61] W. Li, J. Han, and J. Pei. Cmar: accurate and efficient classification based on multiple class-association rules. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 369–376, 2001.

[62] S. Liao, T. Zou, and H. Chang. An association rules and sequential rules based recommendation system. In *Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference on*, pages 1–4, 2008.

[63] Y. J. Lim and Y. W. Teh. Variational bayesian approach to movie rating prediction. In *Proceedings of KDD cup and workshop*, volume 7, pages 15–21, 2007.

[64] C.-J. Lin. Matrix factorization and factorization machines for recommender systems. `https://www.csie.ntu.edu.tw/~cjlin/talks/sdm2015.pdf`, 2015.

[65] W. Lin, S. A. Alvarez, and C. Ruiz. Collaborative recommendation via adaptive association rule mining. In *Proceedings of the International Workshop on Web Mining for E-Commerce (WEBKDD*, 2000.

[66] G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, (1):76–80, 2003.

[67] T.-Y. Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[68] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM, 2011.

[69] L. Martínek. Evaluace algoritmů lokálně senzitivního hashování (lsh) v doporučovacích systémech, 2018.

[70] P. Massa and P. Avesani. Trust-aware collaborative filtering for recommender systems. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 492–508. Springer, 2004.

[71] P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. *Aaai/iaai*, 23:187–192, 2002.

[72] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[73] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa. Effective personalization based on association rule discovery from web usage data. In *Proceedings of the 3rd international workshop on Web information and data management*, WIDM '01, pages 9–15, New York, NY, USA, 2001. ACM.

[74] F. Montagna. An algebraic approach to propositional fuzzy logic. *Journal of Logic, Language and Information*, 9(1):91–124, Jan 2000.

[75] M. Nakagawa and B. Mobasher. Impact of site characteristics on recommendation models based on association rules and sequential patterns. In *Proceedings of the IJCAI*, volume 3, pages 1–10, 2003.

[76] F. Niu, B. Recht, C. Re, and S. J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 693–701, USA, 2011. Curran Associates Inc.

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

[78] Y.-J. Park and A. Tuzhilin. The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 11–18. ACM, 2008.

[79] M. Pavlíček. Doporučovací modely založené na obrázcích, 2018.

[80] I. Pilászy, D. Zibriczky, and D. Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 71–78. ACM, 2010.

[81] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134. ACM, 2002.

[82] B. Recht and C. Re. Parallel stochastic gradient algorithms for large-scale matrix completion. *Math. Program. Comput.*, 5(2):201–226, 2013.

[83] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[84] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, pages 175–186, New York, NY, USA, 1994. ACM.

[85] F. Ricci, L. Rokach, and B. Shapira. Recommender systems: introduction and challenges. In *Recommender systems handbook*, pages 1–34. Springer, 2015.

[86] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[87] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[88] A. Said and A. Bellogín. Comparative recommender system evaluation: benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 129–136. ACM, 2014.

[89] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 791–798, New York, NY, USA, 2007. ACM.

[90] J. J. Sandvig, B. Mobasher, and R. Burke. Robustness of collaborative recommendation based on association rule mining. In *Proceedings of the 2007 ACM conference on Recommender systems*, RecSys '07, pages 105–112, New York, NY, USA, 2007. ACM.

[91] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM conference on Electronic commerce*, EC '00, pages 158–167, New York, NY, USA, 2000. ACM.

[92] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[93] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[94] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28. Citeseer, 2002.

[95] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15 Companion, pages 111–112, New York, NY, USA, 2015. ACM.

[96] G. Shani and A. Gunawardana. Evaluating recommendation systems. In *Recommender systems handbook*, pages 257–297. Springer, 2011.

[97] Y. Shi, M. Larson, and A. Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the Fourth ACM Conference*

on Recommender Systems, RecSys '10, pages 269–272, New York, NY, USA, 2010. ACM.

[98] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[99] Y. Song, A. M. Elkahky, and X. He. Multi-rate deep learning for temporal recommendation. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 909–912. ACM, 2016.

[100] H. Steck. Training and testing of recommender systems on data missing not at random. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 713–722, New York, NY, USA, 2010. ACM.

[101] H. Steck. Item popularity and recommendation accuracy. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 125–132, New York, NY, USA, 2011. ACM.

[102] H. Steck. Evaluation of recommendations: Rating-prediction and ranking. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 213–220, New York, NY, USA, 2013. ACM.

[103] R. Tripodi and S. L. Pira. Analysis of italian word embeddings. *CoRR*, abs/1707.08783, 2017.

[104] V. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264, 1971.

[105] S. Vargas and P. Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 109–116. ACM, 2011.

[106] H. Wang, N. Wang, and D. Yeung. Collaborative deep learning for recommender systems. *CoRR*, abs/1409.2944, 2014.

[107] D. Yarats and A. Bietti. Qmf – a matrix factorization library. `https://github.com/quora/qmf`, 2016.

[108] H. Yin, B. Cui, J. Li, J. Yao, and C. Chen. Challenging the long tail recommendation. *Proceedings of the VLDB Endowment*, 5(9):896–907, 2012.

[109] H.-F. Yu, M. Bilenko, and C.-J. Lin. Selection of negative samples for one-class matrix factorization. In *SDM*, 2017.

[110] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, May 2000.

[111] M. Zanker and M. Jessenitschnig. Case-studies on exploiting explicit customer requirements in recommender systems. *User Modeling and User-Adapted Interaction*, 19(1-2):133–166, 2009.

[112] X.-Z. Zhang. Building personalized recommendation system in e-commerce using association rule-based mining and classification. In *Machine Learning and Cybernetics, 2007 International Conference on*, volume 7, pages 4113–4118, 2007.

[113] R. Zhou, S. Khemmarat, and L. Gao. The impact of youtube recommendation system on video views. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 404–410. ACM, 2010.

[114] Z. Zhu and J. yan Wang. Book recommendation service by improved association rule mining algorithm. In *Machine Learning and Cybernetics, 2007 International Conference on*, volume 7, pages 3864–3869, 2007.

[115] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin. A fast parallel sgd for matrix factorization in shared memory systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 249–256, New York, NY, USA, 2013. ACM.

[116] M. A. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, NIPS'10, pages 2595–2603, USA, 2010. Curran Associates Inc.

Reviewed Publications of the Author Relevant to the Thesis

[A.1] Rehorek T., Kordik P. *A Soft Computing Approach to Knowledge Flow Synthesis and Optimization.* Snášel V., Abraham A., Corchado E. (eds) Soft Computing Models in Industrial and Environmental Applications. Advances in Intelligent Systems and Computing, vol 188., Springer, Berlin, Heidelberg, 2013.

[A.2] T. Rehorek, O. Biza, R. Bartyzal, P. Kordik, I. Povalyaev, O. Podsztavek *Comparing offline and online evaluation results of recommender systems.* REVEAL 2018, a RecSys 2018 workshop, Vancouver, CA, 2018.

[A.3] S. Kuznetsov, P. Kordik, T. Rehorek, J. Dvorak, P. Kroha *Reducing cold start problems in educational recommender systems.* International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, pp. 3143-3149, 2016.

Remaining Publications of the Author Relevant to the Thesis

[A.4] T. Rehorek. *Multiobjective Optimization in Recommender Systems using Ensemble Methods*. Ph.D. Minimum Thesis, Faculty of Information Technology, Prague, Czech Republic, 2013.

Remaining Publications of the Author

[A.5] T. Rehorek, P. Kordik *Using Interactive Evolution for Exploratory Data Analysis.* Proceedings of the 6th International Scientific and Technical Conference (CSIT'2011). Lviv, UA, LPNU, pp. 131–135, ISBN 978-966-2191-04-2. 2011.

Appendix A

Comparing Offline and Online Evaluation Results of Recommender Systems

Comparing Offline and Online Evaluation Results of Recommender Systems

REVEAL Workshop paper

Tomas Rehorek
Czech Technical University, Recombee
tomas.rehorek@recombee.com

Ondrej Biza
Czech Technical University
bizaondr@fit.cvut.cz

Radek Bartyzal
Czech Technical University, Recombee
radek.bartyzal@recombee.com

Pavel Kordik
Czech Technical University, Recombee
kordikp@fit.cvut.cz

Ivan Povalyev
Recombee
ivan.povalyev@recombee.com

Ondrej Podsztavek
Czech Technical University
podszond@fit.cvut.cz

ABSTRACT

Recommender systems are usually trained and evaluated on historical data. Offline evaluation is, however, tricky and offline performance can be an inaccurate predictor of the online performance measured in production due to several reasons. In this paper, we experiment with two offline evaluation strategies and show that even a reasonable and popular strategy can produce results that are not just biased, but also in direct conflict with the true performance obtained in the online evaluation. We investigate offline policy evaluation techniques adapted from reinforcement learning and explain why such techniques fail to produce an unbiased estimate of the online performance in the "watch next" scenario of a large-scale movie recommender system. Finally, we introduce a new evaluation technique based on Jaccard Index and show that it correlates with the online performance.

CCS CONCEPTS

• **Information systems** → **Collaborative filtering**; • **Theory of computation** → *Reinforcement learning*;

KEYWORDS

Recall, CTR, Recommender Systems, Policy Evaluation

ACM Reference Format:
Tomas Rehorek, Ondrej Biza, Radek Bartyzal, Pavel Kordik, Ivan Povalyev, and Ondrej Podsztavek. 2018. Comparing Offline and Online Evaluation Results of Recommender Systems: REVEAL Workshop paper. In *Proceedings of RecSyS conference (RecSyS'18).* ACM, New York, NY, USA, Article 4, 5 pages. https://doi.org/10.475/123_4

RecSyS'18, , Vancouver, Canada

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

1 INTRODUCTION

Online evaluation is the best approach to assessing the performance of recommender systems, but it poses many challenges: deploying models online is time-consuming, models with poor performance harm user experience and the measurements are irreproducible. Therefore, offline evaluation on historical data is often used to train recommender systems and select candidates that might perform well online. The more offline measures correlate with online results the better.

There are many problems preventing offline evaluation methods from being unbiased estimates of the online performance. Many studies [20] have shown that offline measures such as root-mean-square error (RMSE) on historical ratings are poor estimators. Moreover, it is hard to estimate how the user would have reacted if presented with a different set of recommendations. When an evaluated algorithm generates a different recommendation than the algorithm used to collect historical user interactions, the performance estimate is poor.

Recommendation, similarly as search or any other learning-to-rank problem, can also be viewed as a reinforcement learning problem, where selecting good recommendation leads to higher future rewards (clicks, purchases). Although most companies running recommender systems are oriented towards short-term rewards that are easier to measure (e.g., immediate click-through-rate and conversion rate), optimizing long-term rewards such as customer lifetime value leads to lower churn of users, increased satisfaction and loyalty, and pays off in the long-term. In this paper, we also focus on short-term performance criteria, but there is space for further extension of the proposed method to long-term evaluation.

Recommendation as a reinforcement learning problem has been studied in [2, 23, 28]. It is easy to map a recommendation task into the reinforcement learning domain. Actions are possible recommendations for a given user in a given state, rewards can be derived from implicit user ratings and policies are recommendation algorithms. The main problem is that the number of possible actions (ranked lists of recommended items) can be enormous for real-world recommenders (having millions of items that can be recommended alone, not even taking their combinations into account). Even simple bandit-based reinforcement learning algorithms suffer from scalability issues.

Considering a recommendation task simplified to generate a single item, it is possible to imagine a context-free greedy k-bandit such as the "trending bestseller" model that generates recommendations based on the recent global popularity of items. Such an algorithm is not very competitive in most recommendation scenarios, and hence contextual bandit algorithms [21] should be employed instead. Deep learning embeddings [3] can be utilized to process and represent the context (e.g. a sequence of deep embeddings of purchased items for each user) so the challenge is to predict the reward [24] or the Q-function [28].

Instead of designing a good and scalable reinforcement learning algorithm for recommendation, which is still a work in progress, this paper targets another important challenge: evaluating recommendation algorithms properly on offline data.

Sampling methods use selected historical recommendations to reduce bias. Weighted importance sampling [15] can be viewed as a special case of weighting the error of individual training samples. Doubly robust evaluation [5, 10] is useful when there is either a good model of rewards or a good model of past policy.

In recommender systems, there are different probabilities of the user observing particular an item. When these probabilities can be estimated from historical data, the Inverse-Propensity-Scoring (IPS) estimator [19] can compute an unbiased offline score. However, estimating these probabilities in large-scale dynamic environments is neither practical nor easy. Basic IPS estimators can even have a negative correlation with the online performance as measured in [8].

We experiment with classical content-based and collaborative filtering algorithms for recommendation and run large-scale experiments to show how different offline evaluation strategies correlate with the online performance.

2 RELATED WORK

Said et al. evaluated basic recommendation algorithms from three different open-source frameworks on Movielens and Yelp datasets [17]. They measure how various aspects of evaluation, including strategies for data splitting (e.g. cross-validation vs. 80%-20% split) and candidate item generation, affect prediction accuracy (RMSE), ranking quality (nDCG@10), catalog coverage and running times. Their unified evaluation uncovered significant differences in prediction accuracy between different implementations of the same algorithms.

Offline evaluation of Contextual Bandits was studied in [5–7, 9, 11–14, 16, 27]. A replay-based evaluation method was first proposed in [11] and further studied in [12, 13]. The method considers only the logged data that match the recommendations of the evaluated model. [13] proved their evaluator is unbiased given an infinite data stream of i.i.d. events from a uniformly random logging policy. A common trait of replay-based evaluators is that only a fraction of events generate the final score. This can cause the evaluator to be biased towards short sequences of events (because the data stream in never infinite), as discussed in [16], where controlling the bias with bootstrapping techniques is suggested.

A benefit of replay-based methods over simulating the environment is that we can avoid modeling bias. [5] combined a model of the reward function with Importance Sampling to form a Doubly Robust estimator that mitigates the bias introduced by the model and the high variance of Importance Sampling. [10] derived the Doubly Robust estimator for the full reinforcement learning problem and [7, 25, 27] proposed further improvements to the estimator. A lower bound on a return of a trajectory (a sequence of recommendations for a single user) based on Importance Sampling was derived in [26] and compared with online evaluation in [24].

Handling selection bias in the evaluation and training of recommender systems was explored in [19]. The introduced approach is based on Propensity Scoring, where propensities were estimated by Naive Bayes. Results on two datasets indicate that bias is reduced, but the online performance was not measured to confirm the hypothesis.

In [22] offline evaluation for slates recommendation is discussed. The number of possible slates (ranked lists of recommended items) is almost infinite given the number of items in real-world databases. It is also not practical to assume that we can estimate the probability that a particular state is generated given complex recommendation algorithms and a high number of slates.

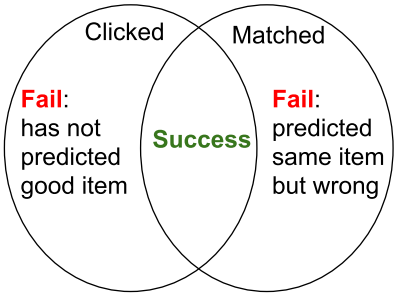

Figure 1: The Jaccard index maximizes the relative size of the region where new algorithm matches the old one on the first recommendation when it was successful (the user has clicked). The other regions are considered a failure. Recommendations that are both unmatched and unclicked are not taken into account, because there is no hint if they can succeed.

3 OUR APPROACH

We argue that all the above-mentioned approaches either do not give us an unbiased estimate of the online performance or come with too strong assumptions that are inappropriate or hardly applicable in production environments. When offline data are generated by standard collaborative filtering based algorithms, most of the assumptions that were used to derive the estimators are violated.

We designed and explored several estimators and found out that one performs particularly well. We extended an IPS estimator (Algorithm 2 from [12]) which is penalizing algorithms similar to the one used to obtain offline data as we explain in the next Section.

Our Jaccard Index based Estimator (JIE) reduces this penalty by normalizing successful hits (number of recommendations with matched first item followed by a click or a conversion) by unsuccessful attempts when the evaluated algorithm a) has not recommended clicked first item successful predicted by online data producer, or b) recommended the same item as online data producer but there was no click or conversion (see Figure 1).

$$JIE = \frac{\text{clicked} \cap \text{matched}}{\text{clicked} \cup \text{matched}}$$

Compared to [13], we do not assume i.i.d. generation process. This has two important consequences. 1) The online performance estimation is biased towards the online generation process, such as the currently deployed collaborative filtering algorithm. Specifically, we are possibly unfairly penalizing models that would have good performance, yet by recommending completely different items. This disadvantage is, however, compensated by 2) There is no need to expose users to random recommendations, significantly damaging their trust in the recommendations and possibly the product image of the whole system. In scenarios like similar/related items recommendation, using a random model is hardly possible.

To compute JIE, we iterate through all recommendations generated by the original model $model_o$ during a selected, recent time period. We denote R_{orig} the set of records containing collected information about these recommendations. Each entry $(user_o, time_o, recom_o, clicked_o) \in R_{orig}$ holds information that $user_o$ has been shown $recom_o$ as the first recommend item at $time_o$ with boolean flag $clicked_o$ determining whether the user was a reward or not. For each record in R_{orig}, we generate alternative recommendation $recom_i$ by all the models from M, simulating exactly the same conditions to those that were present when generating $recom_o$ by production model at $time_o$. This includes hiding all interaction data that appeared after $time_o$ and using the exact same business rules applied to the corresponding recommendation request.

Finally, we compute JIE for all the alternative models $model_i \in M$ by aggregating numbers from cases when there is either match between $recom_o$ and $recom_i$, or when $recom_o$ has been clicked, considering successful only the cases when both match and click happened. See the JIE computation methodology in Alg. 1 below.

Algorithm 1 Jaccard Index based Estimator computation

for $model_i \in M$ **do**
  $success_i \leftarrow 0$
  $clicked_i \leftarrow 0$
end for
for $(user_o, time_o, recom_o, clicked_o) \in R_{orig}$ **do**
  for $model_i \in M$ **do**
    $recom_i \leftarrow model_i(user_o, time_o)$
    $matched_i \leftarrow recom_o = recom_i$
    if $clicked_o \vee matched_i$ **then**
      $total_i \leftarrow total_i + 1$
      if $clicked_o \wedge matched_i$ **then**
        $success_i \leftarrow success_i + 1$
      end if
    end if
  end for
end for
return $\left(\frac{success_i}{total_i}, \ldots, \frac{success_{|M|}}{total_{|M|}}\right)$

4 EXPERIMENTS

An important contribution of our work is that we were able to validate our theoretical hypotheses in a large-scale production environment. We are aware of the limited reproducibility of our results; however, it is hard to reproduce online tests with real users. We believe that our findings are still interesting for the research community and can be reproduced by another team with access to a large-scale recommendation infrastructure.

Our aim was to measure the correlation between the proposed Jaccard Index based Estimator (JIE) and the true online performance (CTR) of candidate models. Our client Showmax agreed with online experiments on a small portion of the "watch next" recommendation scenario to verify the hypotheses discussed in this paper. Henceforth, during the evaluation period, we let the current production model ($model_o$) generate the recommendations for the majority of users, but for a limited subset of users, we deployed individual candidate models $model_i \in M$. These models were evaluated both online (measuring true CTR) and offline by JIE. Thanks to this, we are able to compare the estimation performance of JIE.

One of the challenges is that our customers use a query language (ReQL) on top of each recommendation request, allowing them to filter out or boost particular items and more. It is crucial to exactly emulate ReQL business rules offline to get results comparable to the online behavior of algorithms under investigation. This again complicates reproducibility and generalization of our results to other recommendation scenarios with different dynamics. Nevertheless, we believe our results are valuable with reasonable chance that the hypotheses holding for a particular "watch next" scenario will hold for other scenarios as well.

The model currently used in production is an improved version of Collaborative Filtering User-kNN algorithm with cosine similarity and Non-normalized Cosine Neighborhood as defined in [4], using aggregated implicit ratings. The modification is based on using attribute-based or popularity-based models when there is not enough confidence. But considering the given scenario, data density, and used ReQL business rules, the difference between pure and our modified version of User-kNN is small in the most cases.

The models in M we decided to evaluate were:

- *user-knn* – a pure form of Collaborative Filtering User-kNN algorithm with cosine similarity and Non-normalized Cosine Neighborhood as defined in [4], nearly identical to the one running in production,
- *rating-itemknn* – Item-Based Collaborative Filtering k-Nearest Neighbor with cosine similarity as defined in [18],
- *token-itemknn* – Item-Based k-Nearest Neighbor algorithm as defined in [18], but with significantly different similarity measure $sim(i, j)$ working with item attributes, making the algorithm Content-Based rather then Collaborative Filtering. Specifically, Showmax has a mixture of categoric attributes, tags, and text descriptions, all of which are parsed and converted to a common set of *tokens* for each item. When measuring the similarity between two items i and j, modified Jaccard similarity with a TF-IDF-based weighting of individual tokens is used.

We compare online and offline evaluation results for the three models using two different offline evaluation algorithms. We chose Algorithm 2 from [13] as a baseline. The algorithm assumes the logging policy is uniformly random, which is not the case in our experiments. To correct for the bias introduced by the production

Table 1: Online test results on first recommended item

model name	total recomms	actions	CTR (%)
rating-itemknn	9580	368	3.84
token-itemknn	9829	537	5.46
user-knn	9476	588	6.21
default	168848	10234	6.06

Table 2: Bias in offline results - user-knn similar to default recommendations leading to much more matches but also less percentage of actions

model name	total recomms	actions	CTR (%)
rating-itemknn	9536	352	3.69
token-itemknn	12937	668	5.16
user-knn	109027	1372	1.26

Table 3: Jaccard Index Estimator correlates with the online performance

algorithm	recomms	hits	$match \cup click$	JIE
rating-itemknn	148337	372	16681	0.022
token-itemknn	148340	252	11104	0.022
user-knn	148327	2228	76698	0.029

model, we use the Jaccard Index based Estimator described above as the second evaluation algorithm.

5 RESULTS

The results of the online test match our expectations (Table 1). The performance of the user-knn was not statistically different from the default recommendation policy. Token-based itemknn performed slightly worse and the worst performer was the rating-itemknn.

Table 2 shows offline estimates measured using the Algorithm 2 in [13]. The results are strongly biased, greatly underestimating the performance of the user-knn. The source of the bias is the policy that generated the offline data: instead of a randomly uniform logging policy, the recommendations were generated by an ensemble of the user-knn and other methods. As you can see, the number of matching recommendations for the user-knn is much higher than for the other two algorithms. The problem is that, for rating- and token-itemknn, we only consider recommendations that match the logging policy and in such cases (when two diverse algorithms compromise on the first recommendation) the confidence of the recommendation is high, hence higher CTR and biased estimate.

Table 3 shows offline estimates by our proposed JIE method implemented by Algorithm 1. In the $match \cup click$ column showing denominators of Jaccard similarity, we can see that $token - itemknn$ has much higher overlap with the production algorithm than $rating - itemknn$ on the first recommended item. The biggest with $user - knn$ is orderly larger because the two algorithms are nearly identical.

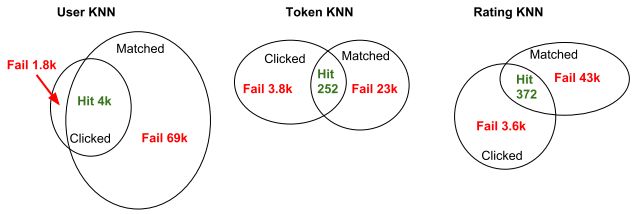

Figure 2: Whereas $userknn$ matches most of the recommendations produced by default policy, the overlap in clicks is also high. For $token - itemknn$ as a different algorithm, the number of matching first items was significantly smaller and a lot of clicks was not predicted. Even worse match and fails in prediction was measured for the $rating - itemknn$.

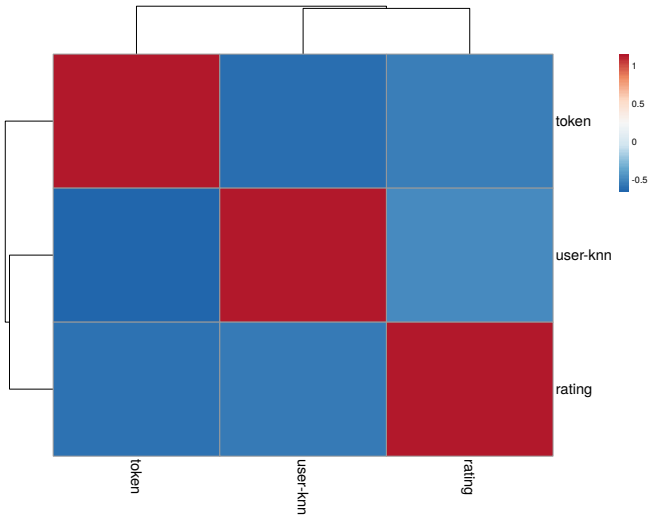

Figure 3: Heatmap of matched recommendation between logging and evaluated policy shows that token and user item-knn share slightly more recommendations than user and rating knn. Proportions of matched recommendations can be used to evaluate diversity of policies.

Similar results can be observed in Figure 2 decomposing JIE to components for an independent evaluation run. Again, offline results correlate with the online performance of the algorithms.

Finally, we decided to run an additional experiment on different recommendation scenario. Contrary to the watch next scenario, the selected scenario presented many items in a row so the visual dominance of first recommended items was absent. We found out, that for this scenario, it is beneficial to compute JIE not just from the single first item, but from K first items displayed to users.

In this scenario, we count match as number of corresponding items between the online and evaluated policy for each recommendation. For match equal to 1, recommendations have to be identical. Zero match means no overlap. We summarized and normalized matches for all recommendations and for all policies in a cross-validation manner.

Figure 3 shows that all three policies are significantly different. The strongest match was always when policies were identical. The number of matched items is also symmetrical following theoretical expectations.

6 DISCUSSION

The proposed offline evaluation methodology and the Jaccard Index Estimator is not completely unbiased.

One possible problem arises when the recommendation algorithm is largely different from the logging policy. The number of matches will be very low in this case and so the number of hits and the confidence of our estimator. We will perform additional experiments to explore such cases.

7 CONCLUSION

Estimation of the online performance from offline data is a difficult task. The main contribution of this paper is that we measured and explained the bias of existing evaluation methods. We showed that the best correlation with the online performance was achieved by Jaccard Index between successful conversions and corresponding recommendations of the evaluated algorithm and the algorithm used to obtain the offline data. Our future work is to investigate the proposed estimator in a much broader experimental setup with hundreds of policies and tens of different recommendation scenarios. We also plan to study how to further improve the robustness of estimates by incorporating propensity scoring where propensities will be estimated by a recently proposed approach [1].

8 ACKNOWLEDGEMENT

This research was partially supported by grant no. GA201/05/0325 of the Grant Agency of the Czech Republic.

REFERENCES

[1] Aman Agarwal, Ivan Zaitsev, and Thorsten Joachims. 2018. Consistent Position Bias Estimation without Online Interventions for Learning-to-Rank. *arXiv preprint arXiv:1806.03555* (2018).

[2] Sungwoon Choi, Heonseok Ha, Uiwon Hwang, Chanju Kim, Jung-Woo Ha, and Sungroh Yoon. 2018. Reinforcement Learning based Recommender System using Biclustering Technique. *arXiv preprint arXiv:1801.05532* (2018).

[3] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. New York, NY, USA.

[4] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of Recommender Algorithms on Top-n Recommendation Tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems (RecSys '10)*. ACM, New York, NY, USA, 39–46. https://doi.org/10.1145/1864708.1864721

[5] Miroslav Dudík, John Langford, and Lihong Li. 2011. Doubly Robust Policy Evaluation and Learning. In *Proceedings of the 28th International Conference on International Conference on Machine Learning (ICML'11)*. Omnipress, USA, 1097–1104. http://dl.acm.org/citation.cfm?id=3104482.3104620

[6] Miroslav DudÃŋk, Dumitru Erhan, John Langford, and Lihong Li. 2014. Doubly Robust Policy Evaluation and Optimization. *Statist. Sci.* 29, 4 (11 2014), 485–511. https://doi.org/10.1214/14-STS500

[7] Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. 2018. More Robust Doubly Robust Off-policy Evaluation. *CoRR* abs/1802.03493 (2018). arXiv:1802.03493 http://arxiv.org/abs/1802.03493

[8] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. 2018. Offline A/B testing for Recommender Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 198–206.

[9] William Hoiles and Mihaela Van Der Schaar. 2016. Bounded Off-policy Evaluation with Missing Data for Course Recommendation and Curriculum Design. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 1596–1604. http://dl.acm.org/citation.cfm?id=3045390.3045559

[10] Nan Jiang and Lihong Li. 2016. Doubly Robust Off-policy Value Evaluation for Reinforcement Learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 652–661. http://dl.acm.org/citation.cfm?id=3045390.3045460

[11] John Langford, Alexander Strehl, and Jennifer Wortman. 2008. Exploration Scavenging. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*. ACM, New York, NY, USA, 528–535. https://doi.org/10.1145/1390156.1390223

[12] Lihong Li, Wei Chu, John Langford, Taesup Moon, and Xuanhui Wang. 2012. An unbiased offline evaluation of contextual bandit algorithms with generalized linear models. In *Proceedings of the Workshop on On-line Trading of Exploration and Exploitation 2*. 19–36.

[13] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. 2011. Unbiased Offline Evaluation of Contextual-bandit-based News Article Recommendation Algorithms. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining (WSDM '11)*. ACM, New York, NY, USA, 297–306. https://doi.org/10.1145/1935826.1935878

[14] Lihong Li, Remi Munos, and Csaba Szepesvari. 2015. Toward Minimax Off-policy Value Estimation, In Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS). https://www.microsoft.com/en-us/research/publication/toward-minimax-off-policy-value-estimation/

[15] A Rupam Mahmood, Hado P van Hasselt, and Richard S Sutton. 2014. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems*. 3014–3022.

[16] Olivier Nicol, Jérémie Mary, and Philippe Preux. 2014. Improving offline evaluation of contextual bandit algorithms via bootstrapping techniques. In *International Conference on Machine Learning (Journal of Machine Learning Research, Workshop and Conference Proceedings; Proceedings of The 31st International Conference on Machine Learning)*, Eric Xing and Tony Jebara (Eds.), Vol. 32. Beijing, China. https://hal.inria.fr/hal-00990840

[17] Alan Said and Alejandro Bellogín. 2014. Comparative recommender system evaluation: benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender systems*. ACM, 129–136.

[18] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International Conference on World Wide Web (WWW '01)*. ACM, New York, NY, USA, 285–295. https://doi.org/10.1145/371920.372071

[19] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations As Treatments: Debiasing Learning and Evaluation. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 1670–1679. http://dl.acm.org/citation.cfm?id=3045390.3045567

[20] Harald Steck. 2011. Item popularity and recommendation accuracy. In *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 125–132.

[21] Richard S. Sutton and Andrew G. Barto. 1998. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks* 16 (1998), 285–286.

[22] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miro Dudik, John Langford, Damien Jose, and Imed Zitouni. 2017. Off-policy evaluation for slate recommendation. In *Advances in Neural Information Processing Systems*. 3632–3642.

[23] Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. 2007. Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM conference on Recommender systems*. ACM, 113–120.

[24] Georgios Theocharous, Philip S Thomas, and Mohammad Ghavamzadeh. 2015. Personalized Ad Recommendation Systems for Life-Time Value Optimization with Guarantees.. In *IJCAI*. 1806–1812.

[25] Philip Thomas and Emma Brunskill. 2016. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*. 2139–2148.

[26] Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. 2015. High-Confidence Off-Policy Evaluation. https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/10042

[27] Yu-Xiang Wang, Alekh Agarwal, and Miro DudÃŋk. 2017. Optimal and Adaptive Off-policy Evaluation in Contextual Bandits, In Proceedings of the 34th International Conference on Machine Learning. 70, 3589–3597. https://www.microsoft.com/en-us/research/publication/optimal-adaptive-off-policy-evaluation-contextual-bandits/

[28] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A Deep Reinforcement Learning Framework for News Recommendation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 167–176.