Benchmarking Recommender Systems

Inauguraldissertation zur Erlangung des akademischen Grades eines
Doktors der Wirtschaftswissenschaften durch die
Wirtschaftswissenschaftliche Fakultät der
Westfälischen Wilhelms-Universität Münster

Vorgelegt von

Dipl.-Inf. Leschek Adam Homann geb. Fitzek
aus Krappitz, Polen

Dezember 2020

Dekan	Prof. Dr. Gottfried Vossen
Erster Gutachter	Prof. Dr. Gottfried Vossen
Zweiter Gutachter	Prof. Dr. Stephan Meisel
Mündliche Prüfung	05.02.2021

Acknowledgements

First and foremost, I would like to thank my doctoral adviser Prof. Dr. Gottfried Vossen. I am grateful for the opportunity to write my thesis in his DBIS Group and his support during the research and writing process. I would also like to thank Prof. Dr. Stephan Meisel for engaging as a second reviewer and Prof. Dr. Sonja Gensler for taking the role of the third reviewer during the defense of this thesis.

Furthermore, I am very grateful to my colleagues at the DBIS Group: Dr. Jens Lechtenbörger and Dr. Denis Martins for providing valuable feedback as well as Felix Nolte, Jan Everding, and Nico Grohmann for participating in my rehearsals for the disputation. I am also thankful to Ralf Farke for his technical support during my research and Julia Seither for her assistance in bureaucratic matters. I thank all of you for your kindness and friendship. Additionally, I would like to thank my former colleagues Dr. Florian Stahl, Dr. Fabian Schomm-von Auenmüller, Dr. David Fekete, and Dr. Nicolas Pflanzl for their support at the start of my employment at the chair.

I also like to thank the members of the Omni-Channel Lab powered by Arvato for the three amazing years of collaboration between the ERCIS and Arvato CRM: Dr. Matthias Carnein for providing valuable input for my thesis and Markus Heuchert for the joint work in the lab, Karsten Kraume, Jiaqing Zhong, and Klaus Voormanns from Arvato for their support throughout the project as well as Prof. Dr. Gottfried Vossen, Prof. Dr. Heike Trautmann, and Prof. Dr. Dr. h.c. Dr. h.c. Jörg Becker.

Furthermore, I would also like to thank Prof. Dr. Bart Goethals from the University of Antwerp for the opportunity to visit his group to discuss and present my research.

I would like to express my gratitude to my parents Irene and Herbert Fitzek for always believing in me and for all the support I received from them in my life. Also, I am grateful to my lovely parents-in-law Monika and Karl-Heinz Homann, who always let me feel as being their own son. Unfortunately, Karl passed away before this thesis was finished, but his spirit and optimism are always around us. I also thank Monika for providing me access to her office and, especially, for the exciting conversations during our lunchtimes, which always were a pleasant distraction.

Finally, I would like to thank my incredible wife Linda Homann for her unlimited support. Without her, this thesis would not be written yet. In times of doubt, her positive thinking helped me to believe in myself and my work to finish this thesis. She did all of this besides being a wonderful and loving mother to our son Leonhard. I am blessed to have both of you in my life.

Hamm, May 2021 Leschek Adam Homann

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Nowadays, users are confronted with a boundless range of items such as services, products, and information. Accordingly, searching for items represents a tedious and time-consuming task for users. Beyond that, today, users are not even willing to bother to search for items actively. Instead, they prefer a more convenient way to “find” items matching their interests and preferences. To do so, recommender systems are heavily applied as active and personalized filters. Consequently, they are an integral part of users’ daily lives. Whether users are watching movies, looking for new job positions, or browsing in an online catalog, a recommender system is running in the background to generate personalized recommendations based on the users’ behavior and feedback. These include, for instance, rating movies, clicking on job positions, or reviewing products.

Besides, a recommender system is also important from a business perspective since it has proven to increase a company’s revenue. This statement is based on the observation that appropriate recommendations can strengthen user engagement and user satisfaction. As a result, users tend to spend more time consuming a service or purchasing more items. Noteworthy, at this point, are popular streaming services such as Netflix and Spotify, as well as famous online retailers such as Amazon or Zalando. The business value of recommender systems for these companies is further underlined by their support of related recommender system conferences such as the RecSys[1] or by initiating competitions such as the famous Netflix Prize aiming to improve recommendation quality [25].

As a result, recommender systems have become an attractive research field to study both from industry and academia alike. In this regard, most research still focuses on improving recommendation quality as far as possible by developing new approaches, improving established ones, or combining them in new ways. The most considered quality aspect of a recommender system is its accuracy. The accuracy reflects how well a prediction for a so far unknown item matches a user’s real preference or whether a list of recommended items contains relevant ones. The former is termed as the prediction task, and the latter as the top-N task with N denoting the number of recommended items in a list. Trying to improve accuracy by combining approaches might even lead to complex systems, which are hard to deploy, such as stated by Amatriain for the winner of the Netflix Prize [Ama13, p. 2]. Besides, most publications focus on evaluating recommender systems’ accuracy based on standard data sets and one specific type of user feedback such as movie ratings. This evaluation process is supported by many frameworks and libraries, such as *LensKit*, *Implicit*, or *Surprise*, which have been developed for and by researchers.

[1]`https://recsys.acm.org/`

However, to further improve recommendations, it is essential to collect as much information as possible about users to infer their preferences correctly. Since companies reach their users by various communication channels such as email, social media, and the web, the information based on these interactions provides additional insights about the users. Depending on the number of provided communication channels and their interplay, the terms cross-channel, multi-channel, and omni-channel are commonly used, where omni-channel represents the highest form of interplay. For instance, in online retailing, it is still common practice to send newsletters regularly. Combining the information about whether a user clicked a particular item in the newsletter with its purchase history might improve the recommendation results. Additionally, social media provide a communication channel for companies to engage with their users by sharing pictures and commenting on information. Incorporating these social media user interactions into a user's profile might lead to a better understanding of his or her preferences.

1.2 Introductory Example

For a better understanding, a retailing scenario is introduced. A fictitious retailer provides brick and mortar stores and operates an online shop to reach its customers without time and location restrictions. To request information about the status of orders, the retailer offers its customers a service hotline. Furthermore, the retailer leverages social media to engage with its customers. This includes posting new product information or product images that customers can like or comment on. Besides, the retailer regularly sends a newsletter to its subscribers. Customers can write reviews about the purchased products to share their experiences with other customers. Another possibility to express their opinion is to rate the products on a classical 1-to-5 scale. The retailer currently generates personalized recommendations deduced by a customer's purchase history as implemented by many of its competitors. To further improve the recommendations, the retailer decides to utilize additional sources such as whether a customer liked a product image, clicked on the newsletter, or searched for a product.

One challenge a retailer faces is to integrate and associate this information with its customers, e.g., matching customers engaging on social media with those in the customer database. The other challenge considers confidence in the information. For instance, is the gained information that a customer liked a product on social media equally important as the information that he or she bought a product. In other words, the confidence in the information differs among the sources and needs to be examined.

Based on this, the retailer specifies requirements for the new recommender system covering different views. The retailer's management aims to increase the Click-through Rate of the recommendations and the number of recommended items from the product catalog. The technical department has to investigate the impact of the additional sources since more data has to be processed compared to the current implementation. This involves guaranteeing a maintainable system with high throughput and low latency utilizing the new data sources. Moreover, the acceptance of customers for the new recommender system has to be evaluated. This includes determining whether the recommendations represent his or her interests and preferences.

The retailer would like to benchmark various recommender systems and different data source aggregations considering these requirements. Based on the results, the retailer can make a profound decision about *what* sources to use and *which* recommender system approach to apply.

1.3 Goals

As explained in Section 1.1, the amount of available information about users and their behavior is still increasing. One reason is attributed to the fact that companies and services utilize multiple ways to reach their users, e.g., phone, email, or social media, which are termed communication channels. In particular, email and social media provide insights about a user's interest in products as part of a newsletter or a new product posted on social media.

This thesis investigates the influence on user recommendations by incorporating user signals from multiple communication channels. To assess this influence, the thesis develops a benchmark concept that mainly focuses on evaluating omni-channel data. At this point, a benchmark concept and a benchmark itself are suitable since they enable a systematic and comprehensive process for evaluation.

The benchmark concept includes introducing and elaborating different aggregation approaches to combine user signals among the communication channels. For instance, the first aggregation approach considers the confidence in user signals by weighting them depending on the communication channel. The second aggregation approach examines the timely sequence of user signals occurring on each communication channel. The proposed aggregation approaches focus on popular and established Collaborative Filtering approaches. Concluding, the influence of the communication channels depending on the applied aggregation method is evaluated from a user, business, and technical perspective, as suggested in [STS+12]. The user perspective considers the recommendation quality to answer *how well do the recommendations cover the user preferences*. From the business perspective, it is beneficial to know *how many products of the catalog are part of the recommendations*. And finally, the technical perspective considers *how long it takes to generate and provide the recommendations to the users*.

This leads to the following research questions:

1. *How to evaluate recommender systems based on omni-channel data?*

2. *How does omni-channel data influence recommendations?*

Therefore, initially, a benchmark concept is elaborated as part of this thesis. Based on this concept, a benchmark prototype is developed. The implementation utilizes established implementations from the following libraries: *LensKit*, *Implicit* and *Surprise*. After that, the benchmark is applied to a real-world data set from online retailing. This also provides useful results about the individual capabilities and limitations of the implementations. Furthermore, the application of the benchmark provides insights into the real-world data.

1.4 Thesis Structure

This thesis comprises six additional chapters besides this introductory chapter, as depicted in Figure 1.1. Chapter 2 introduces the main concepts of recommender systems, such as their application domains, algorithms, and evaluation. In doing so, the reader gets familiar with the concept of the algorithms considered in this thesis. Furthermore, the evaluation process and the related metrics are explained in detail, considering the different perspectives. Chapter 3 provides an overview of the fundamentals of benchmarking by considering their origin and impact on Information Technology (IT) as well as analyzing established benchmark implementations. This chapter intends to provide a deep understanding of how benchmarks are developed and designed considering their types and target scenarios concerning data models, workloads, and metrics.

Based on the gained knowledge and insights from the previous chapters, a benchmark concept for recommender systems is elaborated with the focus on omni-channel data in Chapter 4. For instance, this concept includes the specification of a data model, data aggregation, workloads, and metrics. In Chapter 5, the benchmark concept is implemented as a prototype. The prototypical implementation of the benchmark offers others and third parties the possibility to apply it. Based on a real-world data set from the online retailing domain, the benchmark is applied in Chapter 6.

Then, the thesis concludes with a summary considering the gained insights and an outlook of future work in Chapter 7. Furthermore, the limitations of the algorithms and the benchmark implementation are discussed. These limitations lead to an outlook on how to improve the benchmark in the future further.

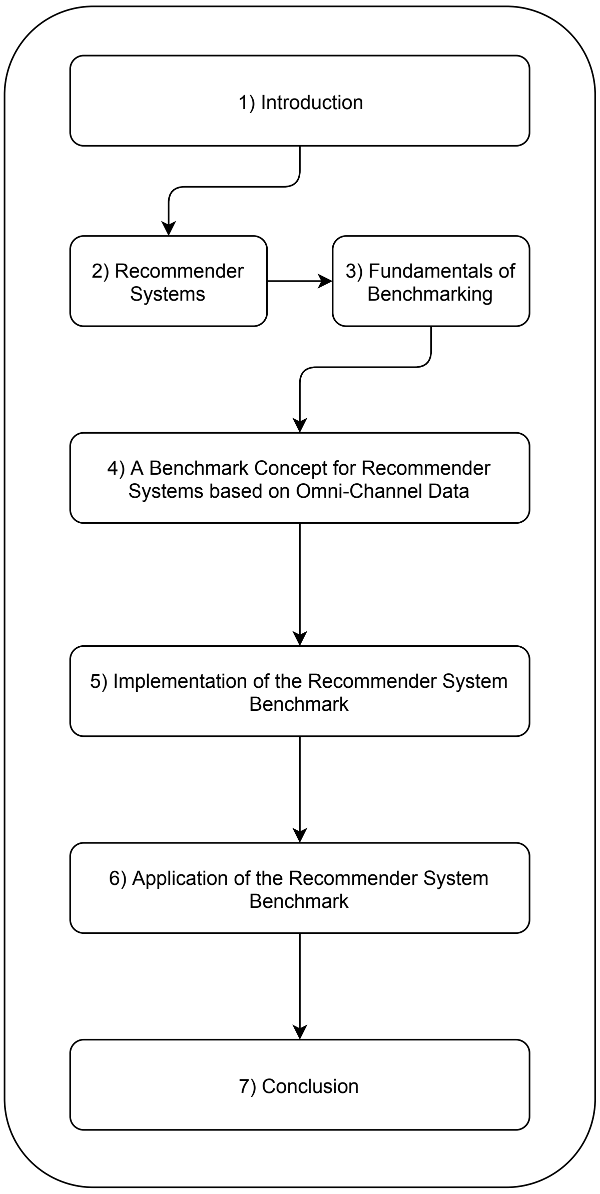

1) Introduction
2) Recommender Systems
3) Fundamentals of Benchmarking
4) A Benchmark Concept for Recommender Systems based on Omni-Channel Data
5) Implementation of the Recommender System Benchmark
6) Application of the Recommender System Benchmark
7) Conclusion

Figure 1.1: An overview of the structure of the thesis. The thesis starts with an introduction containing the motivation and goals of the thesis. Then, in Chapter 2 and Chapter 3, the foundations for the proposed benchmark concept, elaborated in Chapter 4, are given. In Chapter 5, the benchmark implemented is explained. Then, its application is considered in Chapter 6. A conclusion is given in Chapter 7.

2 Recommender Systems

This chapter considers the topic of recommender systems as an essential instrument to provide personalized services to users. Besides the widely known application domain of online retailing, recommender systems are part of professional online networks, social networks, e-dating, tourism services, academic research, and e-learning platforms. Therefore, in the beginning, this chapter elaborates on the purpose of a recommender system and its versatile application domains.

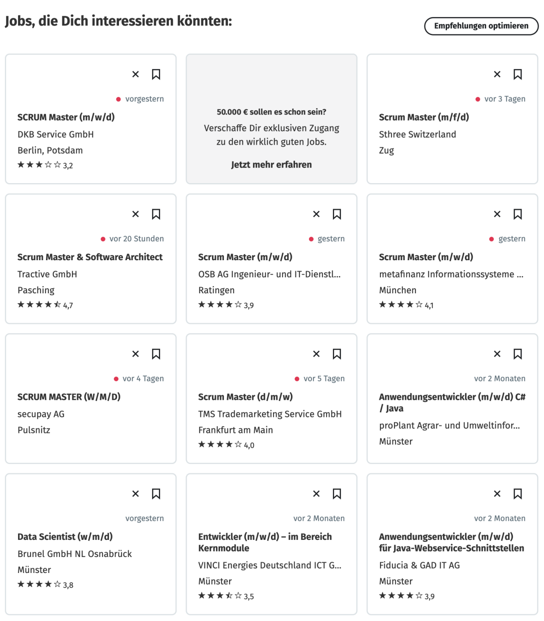
Then, the most popular and established approaches, i.e., Collaborative Filtering and Content-based Filtering, are introduced and explained exemplarily. Subsequently, the evaluation of recommender systems is explained, which involves training and testing techniques as well as metrics for comparative analysis.

This chapter compiles current recommender system software libraries and evaluation frameworks, which simplify the testing and evaluation processes by supporting the automation, the documentation, and the reproducibility of tasks. Furthermore, recommender system frameworks, which provide out-of-the-box solutions, are presented. The chapter concludes with an overview of industrial implementations of recommender systems.

2.1 Motivation

In the past, consumers tended to ask a person of trust, e.g., a family member, a friend, or a familiar salesperson, for recommendations relying on their opinion and experience regarding an item (i.e., a product or a service). For instance, what music album to listen to, what book to read, or what insurance to choose. Some consumers may also read technical literature to collect information about whether an item meets their ideas and requirements to make an appropriate decision. In these scenarios, the recommendations require an explicit action or request from the consumer. In other words, consumers are actively searching for items of interest. Additionally, consumers purchased their items in brick and mortar stores without leaving any personal information regarding their purchase.

With the advent of the Internet, the situation changed considerably for companies and consumers. Nowadays, online retailers, such as Amazon, provide their consumers with a very convenient way to search and buy items without leaving the house and without any restrictions to opening hours. Additionally, the number of available items to choose from has grown considerably since the Internet covers the global market. Even niche items are accessible in an effortless manner. However, the downside of providing a large number of items is that it makes it more difficult for a consumer to find appropriate and interesting ones. This leads to a time-consuming and tedious task for the consumer. Therefore, a system to assist a consumer in this process is desirable.



Jobs, die Dich interessieren könnten:

Empfehlungen optimieren

vorgestern
SCRUM Master (m/w/d)
DKB Service GmbH
Berlin, Potsdam
★★★☆☆ 3,2

50.000 € sollen es schon sein?
Verschaffe Dir exklusiven Zugang zu den wirklich guten Jobs.
Jetzt mehr erfahren

vor 3 Tagen
Scrum Master (m/f/d)
Sthree Switzerland
Zug

vor 20 Stunden
Scrum Master & Software Architect
Tractive GmbH
Pasching
★★★★☆ 4,7

gestern
Scrum Master (m/w/d)
OSB AG Ingenieur- und IT-Dienstl...
Ratingen
★★★★☆ 3,9

gestern
Scrum Master (m/w/d)
metafinanz Informationssysteme ...
München
★★★★☆ 4,1

vor 4 Tagen
SCRUM MASTER (W/M/D)
secupay AG
Pulsnitz

vor 5 Tagen
Scrum Master (d/m/w)
TMS Trademarketing Service GmbH
Frankfurt am Main
★★★★☆ 4,0

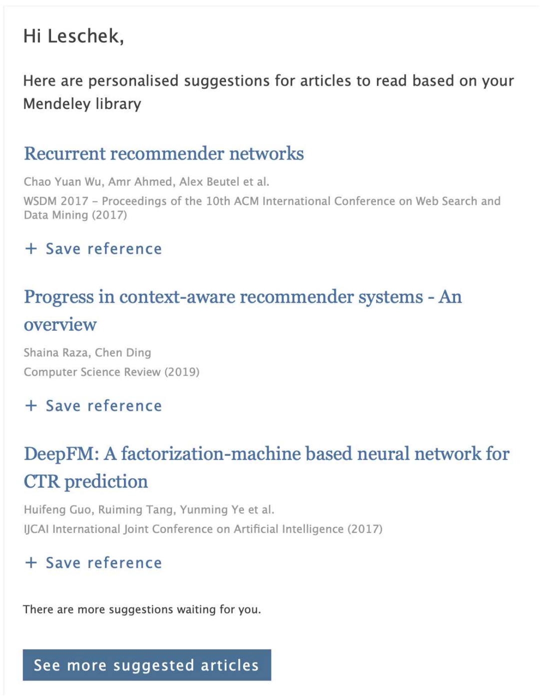
vor 2 Monaten
Anwendungsentwickler (m/w/d) C# / Java
proPlant Agrar- und Umweltinfor...
Münster

vorgestern
Data Scientist (w/m/d)
Brunel GmbH NL Osnabrück
Münster
★★★★☆ 3,8

vor 2 Monaten
Entwickler (m/w/d) – im Bereich Kernmodule
VINCI Energies Deutschland ICT G...
Münster
★★★☆☆ 3,5

vor 2 Monaten
Anwendungsentwickler (m/w/d) für Java-Webservice-Schnittstellen
Fiducia & GAD IT AG
Münster
★★★★☆ 3,9

(a) Job recommendations provided by Xing. Source: [23].



Hi Leschek,

Here are personalised suggestions for articles to read based on your Mendeley library

Recurrent recommender networks
Chao Yuan Wu, Amr Ahmed, Alex Beutel et al.
WSDM 2017 – Proceedings of the 10th ACM International Conference on Web Search and Data Mining (2017)
+ Save reference

Progress in context-aware recommender systems - An overview
Shaina Raza, Chen Ding
Computer Science Review (2019)
+ Save reference

DeepFM: A factorization-machine based neural network for CTR prediction
Huifeng Guo, Ruiming Tang, Yunming Ye et al.
IJCAI International Joint Conference on Artificial Intelligence (2017)
+ Save reference

There are more suggestions waiting for you.

See more suggested articles

(b) Article suggestions provided as weekly newsletter by Mendeley. Source: [48].

Figure 2.1: Recommendations generated in different application domains.

The new possibilities of collecting information about a consumer, e.g., purchases or browsing behavior, enable the generation of personalized recommendations. Consequently, from a business perspective, a well-implemented recommender system supports increasing sales and from a user perspective finding interesting items.

2.2 Application Domains

As discussed in the previous section, a well-known application domain of recommender systems is e-commerce. However, in the last years, recommender systems became an integral part of many application domains, as illustrated in Figure 2.1. In [LWM$^+$15, p. 12], the authors conducted a survey to classify recommender systems by their application domains. Besides e-commerce, they identified the application domains e-government, e-business, e-library, e-learning, e-tourism, e-resources, and e-group activities. However, additionally, the application domains e-dating and e-health are considered. In the following, for each application domain, examples of recommender systems are introduced. The order is based on the respectability of the application domain from the government over the business to the consumer.

E-government

Recommender systems in the e-government application domain aim to support citizens and businesses alike to find the appropriate government services [LWM$^+$15]. The authors further distinguish between Government-to-Business (G2B) and Government-to-Citizen (G2C) recommender systems [LWM$^+$15, p. 16]. For instance, BizSeeker is a G2B recommender system that supports individual businesses in finding a business partner for the distribution of retailing [LWM$^+$15, p. 16]. A G2C recommender system supports citizens to find the right public administration office based on their profile.

E-business

In contrast to [LWM$^+$15, p. 17], which considers e-business recommender systems focusing on providing products and services among businesses, here the term e-business is relaxed to include recommender systems used in a business context from consumers and companies alike.

In this sense, professional online networks provide a serious-minded way to manage a professional business profile. This enables consumers to maintain business relationships as well as to create new business contacts. Another exciting aspect for consumers is to present themselves in the digital job marketplace for other companies and recruiters. The other way around, companies and recruiters can find potential employees meeting their requirements. Therefore, consumers of such platforms groom their personal information seriously and conscientiously. This involves a complete résumé, including information about their educational background, previous employments, and current job position. Besides, the services allow consumers to specify their skills and interests by tags. Other consumers can confirm the correctness of potential skills and interests by endorsements.

Two widely known companies in this domain are LinkedIn[1] and Xing[2]. LinkedIn is the largest platform with about 467 million members [13], whereas Xing is the largest in the German-speaking area with about 15 million members [33]. One of the goals of LinkedIn and Xing is to recommend appropriate jobs to its consumers, as depicted in Figure 2.1a. In this regard, a challenge is the short lifetime of job advertisements, which drives the necessity to provide job recommendations as soon as possible. Furthermore, the services provide recommendations to add additional skills and interests based on similarities to other consumers. Besides, recommendations for potential business contacts are made.

E-health

According to [WP14, p. 2583], the target audience of a recommender system in health care are health professionals and patients. In the process, from a professional's perspective, recommender systems assist in analyzing a large amount of patient data and thereby getting insights, for instance, to predict diseases or to recommend diagnoses, or clinical treatment, whereas a patient might be interested in alternative medicine or a new health insurance. [SPBD19, p. 1]. Besides diagnosis and medication, in [VZV$^+$16, p. 10], the authors mention recommendations for nutrition information or physical exercises. For instance, a recommender system could guide a user to achieve the goal of participating in a marathon by a personalized training plan. Due to the usage of confidential and personal data, such recommender systems have to provide high information quality and trustworthiness as well as to consider authentication and privacy concerns [SPBD19, p. 1].

[1]`www.linkedin.com`
[2]`www.xing.com`

E-library
In academia, researchers rely on software solutions to manage the literature they need for their current paper or are simply interested in. Furthermore, researchers might be interested in collaborations with other researchers working in a similar field of study. Another use case might be to follow specific authors to be aware of new publications.

Mendeley[3] is a widely known and applied tool that leverages information such as the publications in a user's library or the recently read articles to recommend potentially interesting articles to their users. The recommendations are presented in the form "Articles suggested to you related to ...", which complementary delivers an explanation for each recommendation, as shown in Figure 2.1b. Furthermore, people to follow are recommended with explanations such as "Followed by people you follow", "Following the people you follow" or simply since he or she is noted in a specific research field.

E-learning
E-learning, sometimes also referred to as Massive Open Online Courses (MOOC), provides a convenient way for users to improve their skills. In this domain, recommender systems help users to find, for instance, video courses and digital content by their current skills, activities, and experiences [LWM$^+$15, p. 20]. Therefore, this domain is predestined for recommender systems due to the number of available subjects and different skill levels of the users. Providing users with the right and exciting content strengthens his or her confidence in these services. Famous services in this regard are Coursera[4], Udemy[5] or Skillshare[6].

E-tourism
The travel market is still growing and moving from traditional travel agencies to the Internet [39]. Today, nearly two-thirds of all users book their next journey on websites or via apps [39]. In this regard, platforms such as Trivago[7] and Booking.com[8] enjoy high popularity. They provide access to a large number of accommodations, which prevents users from gathering information by themselves. Although it may seem that the platforms just revert to the information given by filters, e.g., price limit or destination, recommender systems are running in the background. Since the platforms' revenue depends on forwardings to hotel websites, it is of high interest for them to provide an adequate ranked list of recommendations [36]. The importance of this is further underlined by a challenge Trivago initiated as part of a recommender systems conference in 2019[9].

E-resource service
As the name suggests, a resource service offers consumers access to a wide range of music, video, audio, and text content. In [LWM$^+$15], content is considered to be user-provided, e.g., by uploads, which is neglected at this point.

[3] `www.mendeley.com`
[4] `https://de.coursera.org/`
[5] `https://www.udemy.com/`
[6] `https://www.skillshare.com/`
[7] `www.trivago.com`
[8] `www.booking.com`
[9] `www.recsyschallenge.com/2019`

Famous companies in this domain are, for instance, Spotify[10], Netflix[11], Audible[12] and, Reddit[13]. Nowadays, such content providers witness increasing popularity on the consumer side. However, this also results in a growing number of competitors. Therefore, companies are interested in increasing the engagement of their consumers with their service. One approach in this context is to provide exclusive content to their consumers, which competitors may not provide. Another approach is to deliver continuously exciting content to their consumers. This leads to a satisfied consumer, which is unlikely to change the provider. Netflix provides most of its recommendations in the form of a so-called matrix-like layout [GUH16, p. 2]. According to a predefined topic, each row in the matrix layout contains an ordered list of recommended videos [GUH16, p. 3-4]. Topics are based on specific movie genres, current trends [GUH16, p. 4] as well as similarities among videos in the form of a "Because You Watched" row [GUH16, p. 4]. Furthermore, Netflix provides recommendations in case a consumer search for a specific movie fails, e.g., because the movie is not available on their platform [GUH16, p. 5]. In such cases, the consumer receives a list of recommendations matching his or her taste.

E-group and social activities
In [LWM+15, p. 24], the authors consider e-group recommender systems to generate recommendations for a group instead of an individual user. In this sense, the preferences and interests of each group member have to be balanced. Nevertheless, here, social networks are also considered part of this application domain since they are based on interactions and relationships among users.

Since its foundation in 2004, Facebook[14] became part of many peoples' daily lives as reflected by nearly 2.4 billion active users per month[10]. Facebook enables users to connect with each other, create and join groups, play games, sell and buy things as well as comment and like content [4]. Companies leverage Facebook to represent their brand, get in touch with potential customers, or promote new products. Also, customer service is provided by some companies due to its convenient application. For instance, airlines use social media to respond to consumer requests about the boarding process, e.g., luggage information, or to support consumers in case of delays, rebookings, or cancellations of flights as discussed in [CHT+17].

Besides Facebook, another popular social network is Twitter[15]. In contrast to Facebook, Twitter provides a more granular partitioning of relationships by allowing users to follow others. Due to this concept and the content limit of 280 characters, Twitter is also termed as a microblogging service. In this regard, users can post messages, so-called tweets, and share messages by retweeting them or mentioning others.

As a consequence, social network providers have detailed information about a user's interests, interactions, and social relationships. Based on this information, social network providers are able to provide recommendations, for instance, about groups a user might be interested in or are popular in his or her area. Furthermore, recommendations of other people

[10]`www.spotify.com`
[11]`www.netflix.com`
[12]`www.audible.com`
[13]`www.reddit.com`
[14]`www.facebook.com`
[15]`www.twitter.com`

a user may know are provided. Additionally, suggestions for interesting videos, pages, and upcoming events are made.

E-dating
Another application domain of high attention for recommender systems is, without any doubt, dating. In 2018, approximately 34 million users in the United States used an online dating platform [12]. This is demonstrated by the fact that the dating app Tinder[16] had the largest reach among all apps in September 2019 [6]. Dating platforms provide an easy way for users to find a partner. In this sense, the usability of Tinder with the concept of swiping right in case a recommended profile attracts a user or swiping left in the other case represents an easy way of interaction. A match between two users is only given when both swiped right, seeing the profile of the other. In this regard, Tinder aims to show profiles as a sequence, which most likely represent a match. As explained by Dr. Steve Liu [47] during his session at the MLconf in 2017, Tinder, besides using the profile information, leverages swipe patterns of their users to provide recommendations.

2.3 Explicit and Implicit Feedback

Recommender systems highly rely on the feedback given by users, which is expressed in an explicit or implicit form. Explicit user feedback involves, for instance, ratings of items on a specified scale, e.g., 1 to 5 to indicate whether he or she considered a movie, song, or book as uninteresting (1) or interesting (5) [JZFF10, p. 22]. Considering online retailing, Amazon represents a good example, where products are rated on a 1 to 5 scale. In the past, Amazon only allowed ratings on products with a corresponding review but changed the concept to a so-called one-click rating system to increase the number of ratings [42]. In this context, a special case is likes or dislikes of items, which can be represented as 0 and 1, respectively. An example in this regard is the swiping concept applied by Tinder or the thumb-up/thumb-down rating system of Netflix [41]. Besides, product reviews are a source of explicit user feedback but are difficult to evaluate since they are represented in unstructured form as text, e.g., comments about hotels. Consequently, this requires the application of Natural Language Processing (NLP) or text mining techniques.

In [AF01, p. 4], Amoo et al. investigate the numeric influence of rating scales on users' responses by comparing scales from 4 to -4 and 9 to 1. Based on their experiments with 139 students, the researchers conclude that negative feedback is considered more negative when it is expressed by negative numbers [AF01, p. 8]. A detailed evaluation of what users expect from a rating scale and the impact of a rating scale are discussed in [CLA$^+$03, p. 586]. In their experiments, the authors applied three different rating scales, thumb-up/thumb-down, a scale from -3 to 3 without the zero and a half star increment scale from 0.5 to 5. One of the experiment outcomes was that most users preferred the fine-grained 0.5 incremental scale [CLA$^+$03, p. 586]. Another outcome was that users should be able to decide which rating scale to use since the user ratings correlated among the different scales. In [GBCV11, p. 128], the authors evaluated six scaling systems: thumb-up/thumb-down, thumb-up/thumb-

[16]www.tinder.com

down/thumb-medium, 3/5/10-points stars and 3/10-points sliders. Their results, however, show no correlation among the different rating scales [GBCV11, p. 130]. This shows the difficulty of choosing the right rating type, e.g., even, odd, or balanced number of ratings and the scale itself.

On the other hand, nowadays, most users often provide feedback unknowingly in implicit form. One reason for this is that the task of rating and reviewing items requires additional effort on a user's side and is perceived as annoying and distracting. To circumvent this situation, implicit feedback aims to infer an interest in or preference for an item leaving a level of uncertainty to the real user preference. Examples of implicit user feedback are purchases of items, consuming media content, or browsing and searching in a product catalog [JZFF10, p. 23], [RRS11, p. 146]. Even though it might sound strange at first to consider a purchase as implicit, the reasoning behind this is that it is not clear whether a user is satisfied with the item. The same holds for pure media consumption, e.g., just watching a movie on Netflix, reading an article on Mendeley, or listening to a song on Spotify. In these cases, the user does not give any explicit feedback about whether the content matches their taste.

Generally, the amount of available implicit user feedback exceeds the amount of explicit user feedback. Even though explicit user feedback is considered as a more trustworthy and reliable source. Nevertheless, both play an important role in modern recommender system implementations, which is discussed elaborately in [BCT18, p. 233-248]

Two common tasks of recommender systems are to generate rating predictions and/or top-N recommendations. The rating prediction task aims to predict a particular rating for an item a user has not rated so far and therefore relies on explicit user feedback. On the other hand, the top-N recommendation task aims to generate a set of N relevant items for a user mostly based on implicit user feedback. Alternatively, a combination of these two tasks can take place by simply using the results of the rating task to generate a sorted list containing the items with the highest prediction ratings as top-N recommendations.

2.4 Approaches

After discussing the main application domains of recommender systems and clarifying user feedback, Collaborative Filtering and Content-based Filtering recommender system approaches are explained. This is based on the fact that they are still the ones witnessing high attention in research and industry alike. In particular, Collaborative Filtering approaches illustrated in Figure 2.2 are considered. Furthermore, this section explains their algorithmic implementations.

2.4.1 Collaborative Filtering Approaches

Collaborative Filtering (CF) relies on the assumption that consumers who shared a similar taste in the past are likely to share a similar taste in the future [JZFF10, p. 13]. In other words, if consumer A, for instance, bought a subset of products consumer B also bought, it seems natural to recommend A some of the products B bought, but A did not. At this point, it must be noted that it requires no additional information about the products themselves to make a feasible recommendation for consumer A. The recommendation is only based on the information that both consumers showed a similar preference pattern, without considering any specific characteristics or features of the products. Extending this approach by considering a community of like-minded consumers leads to the term collaborative. In this sense, for instance, the influence of each consumer inside the community on the overall recommendation could be reflected by the similarity to the target consumer. In this case, the opinion of the most similar consumer for the target product has the highest impact on the final result, followed by the next similar consumer and so forth.

In general CF approaches rely on a utility matrix U, which consists of consumers $C = \{c_1, ..., c_n\}$ represented as rows and items $I = \{i_1, ..., i_m\}$ represented as columns. Here, $n = |C|$ defines the number of consumers and $m = |I|$ the number of items, which leads to a matrix size of $n \times m$. Each cell value $r_{p,j} = U(c_p, i_j)$ indicates an interest of a consumer c_p with $p \in \{1, ..., n\}$ for an item i_j with $j \in \{1, ..., m\}$, as illustrated in Matrix 2.1. Typically, most cell values of the utility matrix U are missing since, for most consumer-item combinations, no indication of interest is available so far. As explained in Section 2.3, an indication of interest is represented explicitly or implicitly.

Generally, CF is divided into two different approaches, namely, memory-based and model-based [JZFF10, p. 26], [BOHG13, p. 113], as illustrated in Figure 2.2. Both approaches utilize the explained utility matrix, representing an indication of interest between a consumer and an item. Memory-based approaches generate recommendations based on the most current and recent information, e.g., a new movie rating or product purchase [BOHG13, p. 113]. To do so, they keep the complete data for the recommendation process in memory and generate recommendations when requested. In contrast, model-based approaches use techniques to process the complete data and transform it. The goal is to deduce a concise and compact model representation, considering the complete data. This, for instance, leads to lower memory consumption. However, a generated model is outdated as soon as new data is available [BOHG13, p. 113].

$$U = \begin{array}{c} \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{array} \begin{array}{c} \begin{array}{cccc} i_1 & i_2 & \cdots & i_m \end{array} \\ \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1m} \\ r_{21} & r_{22} & \cdots & r_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \cdots & r_{nm} \end{pmatrix} \end{array} \tag{2.1}$$

Memory-based

As previously mentioned, CF generates predictions on whether to recommend an item based on similarities among consumers considering their preferences. In this regard, one popular implementation of the memory-based approach determines similar consumers by applying the k-Nearest Neighbor (kNN) method, where k specifies the number of neighbors. In the literature, the implementation is also referred to as user-based CF [JZFF10]. At this point, one widely used way to determine the similarity among consumers c_p and c_q is the Pearson correlation, as explained in [JZFF10, p. 14] and represented in Equation 2.2. Here, the set I contains all items the consumers c_p and c_q showed a preference for. Furthermore, the variables

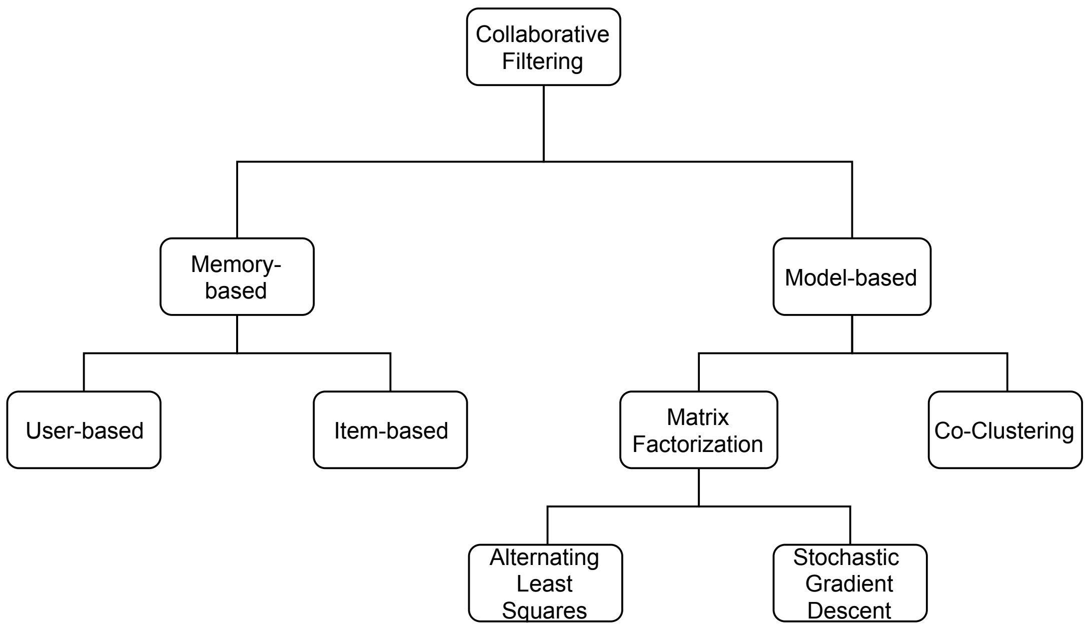


Figure 2.2: An overview of different Collaborative Filtering approaches, based on [BOHG13, p. 112].

$\overline{r_p}$ and $\overline{r_q}$ represent the average consumer preferences of c_p and c_q, respectively. In this case, the numerator of Equation 2.2 adjusts consumer preferences by subtracting their average preferences and multiplying the results. Finally, the denominator of Equation 2.2 normalizes the result to fit into the range of $[-1, ..., 1]$, in which a higher value indicates higher similarity.

$$sim(c_p, c_q) = \frac{\sum_{i \in I}(r_{p,i} - \overline{r_p})(r_{q,i} - \overline{r_q})}{\sqrt{\sum_{i \in I}(r_{p,i} - \overline{r_p})^2}\sqrt{\sum_{i \in I}(r_{q,i} - \overline{r_q})^2}} \tag{2.2}$$

Based on the similarity function, a preference can be estimated, as illustrated in Equation 2.3 for the consumer c_p and item i_j, where N is the set of nearest neighbors determined by the applied similarity function. The sum of the nominator weights the preferences of each neighbor $c_q \in N$ for the target item i_j by its similarity to the target user c_p. Additionally, the average preference of the neighbors is removed to get rid of a consumer's bias. In this context, the bias describes a consumer's tendency to rate low or high generally and, therefore, be considered critical or sweet-tempered. Concluding, the prediction is normalized and adjusted by the average preference of the target consumer.

$$pred(c_p, i_j) = \overline{r_p} + \frac{\sum_{c_q \in N} sim(c_p, c_q)(r_{q,i} - \overline{r_q})}{\sum_{c_q \in N} sim(c_p, c_q)} \tag{2.3}$$

Another implementation of the memory-based approach considers the similarity among items to generate predictions. The concept was published by Amazon in 2003 and termed item-based CF [LSY03]. The basic idea is the following: If two items k and l are similar regarding their consumer preferences and a consumer c_p has already shown a preference for item k, then it makes perfect sense to recommend item l to consumer c_p. In this case, a widely applied function to determine the similarity among item pairs is the adjusted cosine function [JZFF10, p. 19]. To determine the similarity of the items i_k and i_l, as illustrated in Equation 2.4, the set of consumers C only comprises the consumers who have shown a preference for both. The numerator in Equation 2.4 multiplies the preference for the items i_k and i_l for each consumer $c_p \in C$, adjusted by the average preference. The denominator serves to normalize the resulting prediction.

$$sim(i_k, i_l) = \frac{\sum_{c_p \in C}(r_{p,k} - \overline{r_p})(r_{p,l} - \overline{r_p})}{\sqrt{\sum_{c_p \in C}(r_{p,k} - \overline{r_p})^2}\sqrt{\sum_{c_p \in C}(r_{p,l} - \overline{r_p})^2}} \tag{2.4}$$

The prediction for a consumer c_p regarding an item i_k is illustrated by Equation 2.5, where the set RI stands for relevant items and comprises all items the consumer showed a preference for. In the process, for each item i_l in RI, the similarity to the target item is calculated and weighted by a consumer's preference for i_k. In doing so, a high preference strengthens the impact of a similarity.

$$pred(c_p, i_k) = \frac{\sum_{i_l \in RI} sim(i_k, i_l) r_{p,l}}{\sum_{i_l \in RI} sim(i_k, i_l)} \tag{2.5}$$

Implementations of this approach often precompute the similarities of all item pairs to speed up the performance of the recommendation calculation, as described in [LSY03]. On the other hand, the precomputed matrix is out of date as soon as the initial utility matrix is updated since it is based on a snapshot. In this sense, it is a tradeoff between the currentness of data and the cost of generating a recommendation. A representation of a precomputed item-based similarity matrix IB is illustrated in Matrix 2.6. Since the initial utility matrix is processed and transformed into another representation, it is classified as a model-based approach. The same idea is applicable to precompute all consumer pairs, as illustrated by the matrix UB in 2.7.

$$IB = \begin{array}{c} \\ i_1 \\ \vdots \\ i_k \\ \vdots \\ i_m \end{array} \begin{array}{c} \begin{array}{ccccc} i_1 & \dots & i_k & \dots & i_m \end{array} \\ \begin{pmatrix} 1 & \dots & sim(i_1, i_k) & \dots & sim(i_1, i_m) \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ sim(i_k, i_1) & \dots & 1 & \dots & sim(i_k, i_m) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ sim(i_m, i_1) & \dots & sim(i_m, i_k) & \dots & 1 \end{pmatrix} \end{array} \tag{2.6}$$

$$UB = \begin{array}{c} \\ c_1 \\ \vdots \\ c_p \\ \vdots \\ c_n \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & \dots & c_p & \dots & c_n \end{array} \\ \begin{pmatrix} 1 & \dots & sim(c_1, c_p) & \dots & sim(c_1, c_n) \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ sim(c_p, c_1) & \dots & 1 & \dots & sim(c_p, c_n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ sim(c_n, c_1) & \dots & sim(c_n, c_p) & \dots & 1 \end{pmatrix} \end{array} \tag{2.7}$$

Because most platforms provide more items than consumers, the item-based approach demands less memory to store the matrix. Nevertheless, in both cases, the computational effort to precompute the similarity matrices is relatively high. In user-based CF, it is quadratic in the number of consumers, and in item-based CF, quadratic in the number of items.

Model-based

The idea of model-based approaches is to process or transform the initial utility matrix into a model from which preference predictions can be generated [BOHG13, p. 113]. In the following, widely applied approaches in this regard as Matrix Factorization (MF) and Co-Clustering (CoC) are explained.

Matrix Factorization In the context of recommender systems, MF aims to decompose a utility matrix U into matrices of smaller dimension to reveal so-called latent features characterizing consumers and items [BCT18, p. 35]. Here, the features are represented as row and column vectors of the decomposed matrices. The dimension of the vectors is a predefined value, which is identical for the consumer and item features. Thus, only the most important features are considered in the further process.

One approach to decompose a utility matrix $U \in \mathbb{R}^{m \times n}$ is Singular Value Decomposition (SVD) [EY36]. The result of SVD are the three matrices $V \in \mathbb{R}^{M \times M}$, $\Sigma \in \mathbb{R}^{M \times N}$ and $W \in \mathbb{R}^{N \times N}$, as depicted in Equation 2.8.

$$A = V \Sigma W^T \tag{2.8}$$

Here, Σ contains the so-called singular values on its diagonal in ascending order, which describe the impact of the corresponding dimension. The matrices V and W are characterizing the consumer and items, respectively. By considering only r singular values of Σ it is possible to reduce the dimensions of the matrices V and W, which leads to an approximation of A as $V_r \Sigma_r W_r^T$. In other words, the features are projected into an r-dimensional space. After the projection, it is possible to predict preferences by, for instance, applying the cosine similarity among consumers or consumers and items, as illustrated exemplarily in [JZFF10, p. 28]. Unfortunately, SVD can only be applied to dense matrices, which means matrices without missing values [BCT18, p. 41]. In this respect, SVD cannot be used directly on a utility matrix, as illustrated in Equation 2.1, since most of the entries are not set due to the fact that not all consumers interacted with all available items. There exist multiple ways to circumvent this issue, e.g., replacing the missing values with zeros [BCT18, p. 41]. But, this may bias the recommendations to a certain extent. For instance, replacing a missing value with zero may lead to the wrong impression that a consumer is not interested in an item.

Another approach to decompose the utility matrix is to consider it as a minimization problem as represented in Equation 2.9. The idea is to determine feature vectors $p_c \in \mathbb{R}^f$ and $q_i \in \mathbb{R}^f$ for consumers and items such that the dot product $p_c^T q_i$ denoted as $\hat{r}_{c,i}$ approximates a known rating $r_{c,i}$ of consumer c for item i [BCT18, p. 38-39]. Similarly to SVD, f is a predefined value, which describes the dimension of the vectors. Then, these vectors can be used to generate predictions for unknown consumer-item pairs.

In this regard, the task is the following: Given a set of ratings $r_{c,i} \in R$, find the vectors p_c and q_i, which minimize the quadratic error of $r_{c,i} - \hat{r}_{c,i}$ [HKV08]. Since the number of ratings is quite sparse, Equation 2.9 additionally contains a regulation parameter λ to prevent overfitting. In other words, the regulation parameter prevents the vectors from adapting too heavily to the actual ratings.

$$\min_{p*,q*} \sum_{r_{c,i} \in R} (r_{c,i} - p_c^T q_i)^2 + \lambda(\|p_c\|^2 + \|q_i\|^2) \tag{2.9}$$

A further extension of Equation 2.9 is represented in Equation 2.10. Here, the actual ratings of a consumer for an item are adjusted by the overall average rating μ, the average consumer rating b_c and the average item rating b_i [BCT18, p. 43]. The idea is to eliminate bias introduced by consumers and items during the process [BCT18, p. 44]. For instance, some consumers might tend to rate items rather low or high generally. On the other hand, there also exist items that are generally rated lower or higher than others.

$$\min_{p*,q*,b*} \sum_{r_{c,i} \in R} (r_{c,i} - \mu - b_c - b_i - p_c^T q_i)^2 + \lambda(\|p_c\|^2 + \|q_i\|^2 + b_c^2 + b_i^2) \tag{2.10}$$

Until now, all introduced approaches are based on explicit feedback from a consumer for an item. This might not always be the case in real-world scenarios, as discussed in Section 2.3. For instance, a consumer can watch a movie without providing a rating for it at the end. Furthermore, buying an item does not express satisfaction explicitly. To cover this aspect, in [HKV08], the authors introduce the concept of confidence, as illustrated in Equation 2.11.

$$\min_{p*,q*} \sum_{c,i} c_{c,i}(p_{c,i} - p_c^T q_i)^2 + \lambda(\sum_c \|p_c\|^2 + \sum_i \|q_i\|^2) \tag{2.11}$$

Compared to Equation 2.9 the ratings $r_{c,i}$ are replaced by $p_{c,i}$, where $p_{c,i}$ is equal to 1 when $r_{c,i} > 0$ and 0 otherwise. The confidence factor is defined as $c_{c,i} = 1 + \alpha r_{c,i}$. The idea of the confidence factor is to introduce a weighting α, which increases its value, the more confident we are about the user's preference. For example, a user might purchase multiple copies of the same item or browse through a website of the same product several times [HKV08, p. 4]. Additionally, in contrast to Equation 2.9, all missing user-item combinations are considered with a preference of 0. However, these entries are considered with low confidence since no reasonable conclusion about a preference can be inferred from them [KBV09, p. 3]. A downside of the approach is an increased computational effort since all consumer-item combinations are part of the minimization problem.

To solve this kind of minimization problem, two techniques are widely applied, namely, Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS) [BCT18, p. 50-53]. The basic idea of SGD is to adapt the feature vectors p_c and q_i iteratively for each rating $r_{c,i}$. Therefore, SGD first determines the error $e_{c,i}$ between the actual $r_{c,i}$ and predicted rating $\hat{r}_{c,i} = p_c^T q_i$. Afterward, the vectors are updated as represented in Equation 2.12 and 2.13. Here, γ is the learning rate, which specifies the size of movement towards the gradient. To solve the minimization problem of Equation 2.10 the average consumer and average item biases have to updated, as illustrated in Equation 2.14 and 2.15.

$$p_c \leftarrow p_c + \gamma(e_{c,i} q_i - \lambda p_c) \tag{2.12}$$

$$q_i \leftarrow q_i + \gamma(e_{c,i}p_c - \lambda q_i) \tag{2.13}$$

$$b_c \leftarrow b_c + \gamma(e_{c,i} - \lambda b_c) \tag{2.14}$$

$$b_i \leftarrow b_i + \gamma(e_{c,i} - \lambda b_i) \tag{2.15}$$

Alternatively, ALS can be used to solve the minimization problem in Equation 2.10. In contrast to SGD, ALS does not consider each rating separately. Instead, ALS takes advantage of the following observation: Fixing the values of p_c enables to optimize the values of q_i and the other way around. The resulting minimization problems can be solved directly by the least squares approach. The alternating calculation is repeated either by a predefined number of iterations, so-called epochs, or until it converges to a certain threshold. According to [KBV09], due to the independent determination of p_c and q_i, the implementation of the algorithm allows high parallelization. In this regard, it is particularly suited to handle implicit consumer feedback since it has the ability to incorporate all consumer-item combinations in its computation.

Co-clustering In general, clustering is an unsupervised learning technique that aims to separate data into so-called clusters. Data is considered similar within a cluster and dissimilar among different clusters, given a specific characteristic [Liu09, p. 133]. Clustering is classified as unsupervised learning since it requires no prior knowledge about the data for its execution.

One widely applied algorithm for clustering is k-Means [Liu09, p. 136-138]. k-Means works as follows: From a given set of data points, it first selects randomly k data points, representing the initial centroids of the clusters. Then, for each data point, its distance to each centroid is determined and assigned to the cluster with the shortest distance. After assigning each data point to a cluster, the centers of the clusters represent the new centroids. The process is repeated until it converges, i.e., until the cluster assignment stops changing over the iterations.

CoC, sometimes also referred to as biclustering, extends the idea of k-Means. It aims to cluster a matrix by its rows and columns simultaneously [GM05]. In the context of recommender systems, the resulting clusters comprise similar consumers and items. Here, the consumer clusters are denoted as C_u and the item clusters as C_i. The number of clusters is predefined as k_u for consumers and k_i for items, respectively. Based on these clusters, the co-clusters $C_{c,i}$ are determined by relating the corresponding consumer and item clusters. Consequently, this leads to $k_c \cdot k_i$ co-clusters. Besides, CoC is particularly suitable for sparse matrices. The prediction $\hat{r}_{c,i}$ of consumer c for item i is determined by Equation 2.16.

$$\hat{r}_{c,i} = \overline{C}_{c,i} + (\overline{r}_c - \overline{C}_c) + (\overline{r}_i - \overline{C}_i) \tag{2.16}$$

In Equation 2.16, $\overline{C}_c$ and $\overline{C}_i$ are the average consumer and item preferences within the corresponding cluster. $\overline{C}_{c,i}$ is the average rating within a co-cluster. The average consumer and item preferences are denoted as $\overline{r}_c$ and $\overline{r}_i$.

Limitations Despite its popularity, CF comes with some drawbacks and problems. One famous problem in this regard is the cold start problem. The cold start problem addresses the challenge of generating recommendations for an unknown consumer or a new item. In case an unknown consumer starts its interaction with a service, no historical information about its preferences is available. Therefore, CF approaches are not able to determine consumers sharing the same taste based on their previous preference patterns. The same problem occurs when a new item is becoming part of the product catalog. In this case, the item simply disappears since it shares no similarities to other items based on the preference behavior of the consumers.

Another problem results from the sparsity of the utility matrix. In many cases, the utility matrix dimensions are quite high compared to the number of actual preferences represented by it. Therefore, the number of items preferred by two distinct consumers might be quite small. In other words, a consumer shares hardly any similarities to other consumers due to their special preferences. This leads to inadequate and improper recommendations. In the literature, such a consumer is referred to as a grey sheep [GBB16].

In order to solve the cold start problem, different approaches are developed and implemented. One approach to request initial preferences is to present a random but diverse selection of items to an unknown consumer. Based on the ratings of the presented items, the recommender system is then able to generate recommendations.

2.4.2 Content-Based Filtering Approaches

Another widely applied approach in the field of recommender systems is Content-based Filtering (CB) [RRS11, p. 74-100]. In contrast to CF, where recommendations are made based on the taste of similar consumers, CB recommends items that have the same or similar characteristics as the ones a consumer preferred in the past [JZFF10, p. 58]. To do this, CB highly relies on an appropriate consumer profile or model as well as complete item descriptions [CJSD08, p. 7]. In this context, a consumer profile or model reflects the consumer preferences and is generated by applying Machine Learning (ML) techniques [CJSD08, p. 5]. Consequently, a consumer model has to be created initially and updated continuously when new consumer feedback is available.

To exploit similarities between items, CB relies on item descriptions or profiles, which typically consist of metadata represented in structured and unstructured forms. For instance, some metadata values might have constraints such as number ranges or predefined formats, whereas others contain free text. Therefore, item descriptions have to be transformed into a suitable representation, e.g., feature vectors, to check whether an item matches the consumer preferences described by the model. This applies in particular to textual values such as the content of an article or a movie synopsis. At this point, domain knowledge is also beneficial since it enables to leverage experiences regarding item descriptions.

The following simple example illustrates the basic idea of CB. Consider a consumer A who is highly interested in political articles. By doing so, its profile is updated continuously according to the information provided by the articles. Here, this information includes metadata such as the title, the author, the publisher, the release date, the keywords, and the content. In its simplest form, the consumer profile of A maintains a data structure, e.g., a feature vector, containing the same metadata. In order to predict a recommendation for an unread article, it has to be checked whether it reflects the consumer profile. For instance, this means comparing whether the author of the unread article appears in the list of already read authors. Another possibility is to check if the keywords of the unread article are part of the consumer profile. For an accurate prediction, the entire item feature vector is compared against the consumer profile.

Term Frequency - Inverse Document Frequency

Term Frequency - Inverse Document Frequency (TF-IDF) is a technique adopted from Information Retrieval (IR) to represent documents as multidimensional vectors [JZFF10, p. 55]. To do this, irrelevant terms, namely, stop words, such as prepositions and articles, are first removed [JZFF10, p. 56]. Furthermore, it is common to shorten words to their word stem, e.g., replace "recommending" with "recommend". As the name implies, Term Frequency (TF) describes the frequency of a term within a document. On the other hand, Inverse Document Frequency (IDF) denotes the number of documents a term occurs within the complete set of documents, commonly termed corpus. A term that frequently appears in different documents of the corpus is not considered to have high value as a keyword since it does not discriminate the documents quite well. Therefore, the inverse is taken. Here, the formal definition of TF-IDF from [RRS11, p. 82] is used:

$$\text{TF-IDF}(t_k, d_j) = \text{TF}(t_k, d_j) \cdot \log \frac{N}{n_k} \tag{2.17}$$

Here, d_j denotes a document from the corpus $D = \{d_1, ..., d_N\}$, where N is the size of the corpus and n_k the number of occurrences of t_k within the documents. Furthermore, t_k represents a term from the set of distinct terms $T = \{t_1, ..., t_n\}$ occurring in the corpus.

$$\text{TF}(t_k, d_j) = \frac{f_{k,j}}{\max_z f_{z,j}} \tag{2.18}$$

Equation 2.18 defines the calculation of the TF. At is point, $f_{k,j}$ denotes the frequency of t_k in document d_j. Additionally, $f_{k,j}$ is normalized by the most frequently occurring term in document d_j. In doing so, a keyword, which occurs more often in a document compared to other keywords, has higher importance. Finally, the weights $w_{k,j}$ for the document d_j are normalized, as illustrated in Equation 2.19.

$$w_{k,j} = \frac{\text{TF-IDF}(t_k, d_j)}{\sqrt{\sum_{s=1}^{|T|} \text{TF-IDF}(t_k, d_j)^2}} \tag{2.19}$$

Accordingly, a document d_j is encoded as $\{w_{1,j}, ..., w_{n,j}\}$. A widely used measure to compare the similarity between two documents is the cosine similarity [RRS11, p. 82].

Limitations There are three main limitations of this approach, namely, shallow content analysis, overspecialization, and rating acquirement. Lops et al. [RRS11, p. 78-79] mention the same drawback as limited content analysis, overspecialization, and new users. Shallow content analysis refers to extracting appropriate features from texts, images as well as audio and video content. Overspecialization reflects the problem of receiving recommendations containing items that are very similar to those a user already bought in the past. Rating acquirement is related to the cold start problem in CF. A CB recommender system can only provide recommendations when a certain amount of information about a user's preferences is given.

2.4.3 Hybrid Approaches

Hybrid recommender systems aim to balance the limitations of CF and CB approaches by combining them. Considered separately, each approach incorporates only a specific aspect of the available information into the recommendation process. CF relies on the preferences of like-minded consumers within a community, whereas CB utilizes item features and characteristics [JZFF10, p. 124]. Therefore, it seems natural to assume better recommendation results if both approaches are combined to a hybrid recommender system. In this regard, Burke proposes seven ways to implement a hybrid recommender system [Bur02, p. 6-8]. The possibilities are named as follows: weighted, switching, mixed, feature combination, cascade, feature augmentation, and meta-level.

As the name implies, the weighted approach combines different recommendation results to one overall result. In the switching approach, one recommendation technique is preferred to another given a specific criterion. A mixed hybrid system shows the recommendation results determined by multiple techniques. The idea of feature combination is to enrich and enhance content-based information with collaborative information. Both cascade and feature augmentation are built on a sequence of recommendation approaches. In the latter case, the output of one recommender is used as input for the recommender in the process, whereas in the former case, an additional recommender only needs to be applied to discriminate further and refine the recommendations. Finally, a meta-level system goes even one step further by utilizing a learned model from one recommendation approach as input for the upcoming one.

2.4.4 Selected Recommender System Research

A general overview of recommender systems is given by Jannach in [JZFF10]. A detailed survey on recommender systems is elaborated in [BOHG13]. In [BCT18], the authors mainly target the topic of CF recommendations. CB recommender systems are covered by Pazzani and Billsus in [PB07] and Lops et al. in [RRS11, p. 73-100]. A survey of hybrid recommender systems is given by Burke in [Bur02]. Additionally, Burke provides a detailed comparison of different hybrid recommender systems considering the combination of CF, CB, and knowledge-based recommendation approaches in [Bur07]. The authors of [PKCK12] provide a systematic literature review about recommender systems and future directions.

In [SKKR01] item-based CF approaches are analyzed considering item similarity and prediction computation. A further comparison of CF approaches applied to the Netflix and MovieLens data set can be found in [CCFF11]. The application of the item-based approach in a real-life environment is shown by Amazon in [LSY03]. In their paper [DK04], Deshpande et al. investigate top-N item-based algorithms focusing on different similarity measures as well as the generation of recommendations and comparing them with user-based algorithms. The work of [Aio13] particularly focuses on binary implicit feedback to generate top-N recommendations, which was applied by the online retailer Zalando in the past [Fre17]. An approach to incrementally update item similarities in a dynamic environment based on implicit user feedback is proposed in [JVG19]. The developers of TenentRec [HCZ$^+$15] introduce another fast incremental item-based approach, which divides the similarity calculation into three parts for parallelization.

In [DDGR07], Google explains how its news recommender system is implemented as a model-based CF approach based on clustering techniques.

MF and its successful application in the famous Netflix Prize is discussed in [KBV09]. In [TPNT09], further variations of MF approaches are introduced and compared. An incremental MF approach is introduced in [YMJ$^+$16]. Another fast MF approach by He et al. [HPV16] introduces the concept of weighting missing values.

In the industrial context, hybrid recommender systems are widely applied. As explained in [BCT18, p. 571-598], Xing implemented a hybrid recommender system based on CF and CB approaches. In particular, Xing leverage CB to generate user profiles and interest profiles based on TF-IDF. Since the Mendeley recommender system relies on content information [BCT18, p. 604], its implementation is also based on a CF and CB approach. As explained by Freno in [Fre17], CB approaches are part of the Zalando recommender system. The TenentRec [HCZ$^+$15] recommender system combines multiple approaches such as CF, CB, and rule-based. Considering the news domain, Karimi et al. [KJJ18] classify recommender systems into CF, CB and hybrid implementations. One example in this regard is a sport recommender system for the German sports channel Sport1, which is based on a weighting hybrid combination of CB and CF [LH16].

Besides CF and CB approaches, further extensions and approaches are mentioned in the literature. In [JZFF10], for instance, **Knowledge-based Recommender Systems** are described based on the work of [Bur00] and [FB08] and divided into constraint-based and case-based types. Both types try to find appropriate recommendations in an interactive process [JZFF10], utilizing "detailed knowledge about item characteristics" [JZFF10, p. 82] and specific user requirements. In contrast to case-based, which relies on domain knowledge, constraint-based applies filtering mechanisms [FB08]. Considering the computer domain, these requirements or constraints may involve a price range, screen resolution, frame rate, or disk size.

Context-aware Recommender Systems, also termed CARS, aim to include contextual information into the recommendation process. Examples of context information are time, location, weather, activity [BCT18, p. 174-175] or device. As discussed in [BCT18, p. 179], in CF, it is possible to incorporate context as contextual pre- or post-processing as well as contextual modeling. In this regard, CARS can extent other approaches and also operate on their own. Further information about CARS is provided by Adomavicius and Tuzhilin in [RRS11]. For instance, the location and weather might be important to exclude certain places of interest during a city trip since they are too far away from the current user position or not worth visiting due to bad weather conditions. Recommender systems focussing on temporal information are termed **Time-aware Recommender Systems (TARS)** and discussed in [CDC14]. For instance, movie recommendations shown to a user might differ depending on the time, recommending rather entertaining comedy shows in the evening and demanding documentations in the afternoon. Additionally, over time user preferences might change, such that the most recent actives should be considered with higher importance. Another aspect might be how to generate recommendations for a shared account if a household shares one streaming or shopping account. Verstrepen and Goethals address this challenge in [VG15].

Sequence-based Recommender Systems are discussed in [QCJ18]. Here, the authors distinguish between last-N interactions-based, session-based, and session-aware recommendations. Last-N interactions considers only the last n actions of a user. In session-based recommendation, only actions of the last user session are available, whereas, in the session-aware case, past user sessions are available [QCJ18, p. 66]. Generally, sequence-based recommendations rely on time-stamped data collected as a list of user interactions such as viewing, collecting, or purchasing items [BCT18, p. 246]. Depending on whether the current user is known or completely anonymous different information is available. In case the user is known, since he or she is logged into the system or tracked by cookies, recommendations are made by considering their behavior in the past session. For anonymous users, only the information of the current session is available, and recommendations are made based on similar sessions of other users, which is the basic goal, as discussed in [QCJ18].

2.4.5 Recommender Systems as Machine Learning Systems

Machine Learning (ML) combines the fields of computer science and statistics. According to Michael I. Jordan [JM15], the aim of ML is defined as the capability of a system to improve constantly through experience. In this sense, experience means being trained to solve a specific problem to improve a performance measure. Generally, most applications in ML try to improve the performance measure of a function. Such a function can be determined by a search, a factorization, an optimization, or a simulation. Considering this, recommender systems represent a practical application of ML since their goal is to recommend exciting items for users utilizing a function trained on the users' past behavior. Furthermore, recommender systems need to adapt to new data and improve the recommendation results by incorporating this new experience. To measure improvement and thereby learning the system, the specified function is optimized against a predefined metric. In this context, metrics are mostly considering the accuracy of the ML task, such as the correct prediction of ratings or the order of recommendations provided to a user in the recommender system case. These metrics are discussed in the next section.

2.5 Metrics

To compare and analyze recommender systems and their generated results, it must be decided what objective(s) to consider and, consequently, which metrics to apply in the evaluation process. A quote addressed to the famous business pioneer Peter Drucker[17] underlines the importance: "If you can't measure it, you can't improve it". The most applied metrics regarding recommender systems are related to accuracy. Besides, current research also investigates so-called non-accuracy metrics [BCT18, p. 302]. In addition, metrics to develop, deploy,

[17]`https://en.wikipedia.org/wiki/Peter_Drucker`

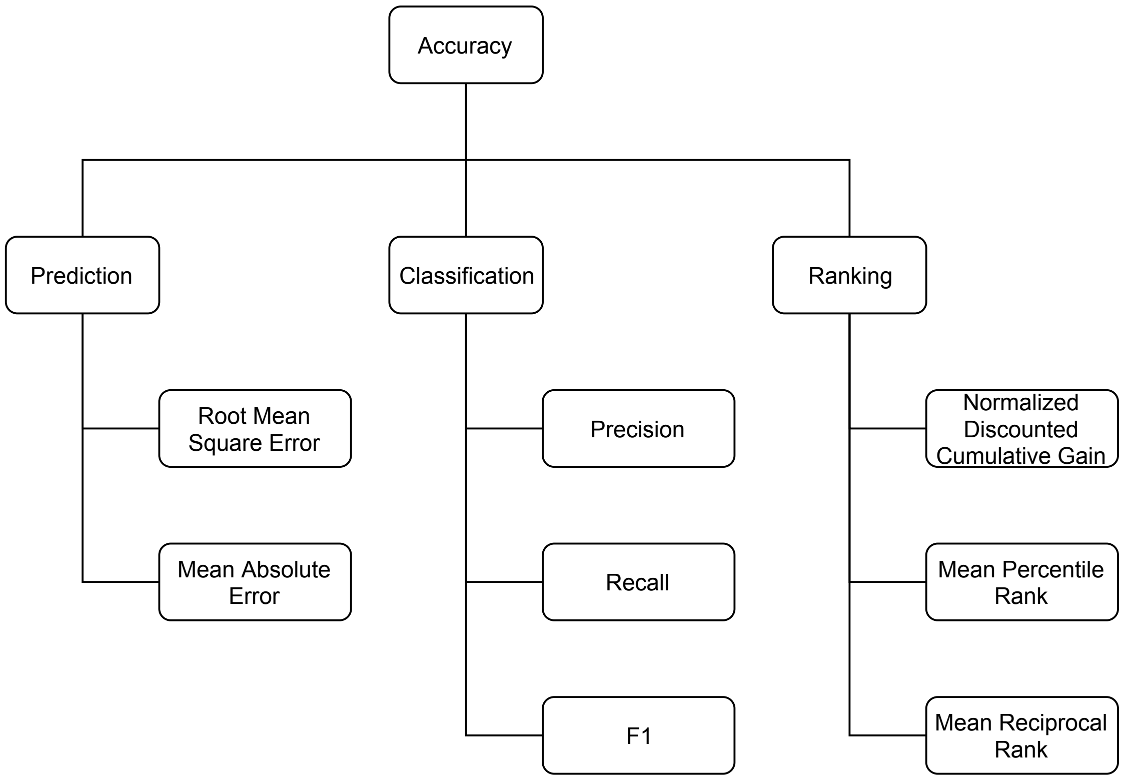


Figure 2.3: Overview of accuracy metrics.

maintain, and run a recommender system are important measurements to consider [BCT18, p. 303], [JZFF10, p. 169]. These metrics are termed performance metrics in the following. Next, different metrics are discussed.

2.5.1 Accuracy Metrics

As illustrated in Figure 2.3, to measure the accuracy of the generated recommendations various metrics exist, which are commonly divided into three classes considering the accuracy of predictions [JZFF10, p. 179] [RRS11, p. 273], the accuracy of classification [JZFF10, p. 180-181] or usage [RRS11, p. 273] and the accuracy of rankings [JZFF10, p. 182] [RRS11, p. 274]. In [BCT18, p. 301], the accuracy measures are considered as error metrics or precision-oriented metrics.

Prediction Accuracy

In the past, recommender systems were mostly evaluated on data sets containing explicit user feedback for items. A famous example in this context is the Netflix challenge, where the user feedback consisted of movie ratings on a scale from 1 to 5. Under this setting, the recommender system's task was to predict the ratings for movies a user had not seen so far. The most common metrics in this regard are the Root Mean Square Error (RMSE) and the Mean Absolute Error (MAE). In the following, R comprises a set of rating predictions $r_{u,i}$ of user u for item i and $\hat{r}_{u,i}$ the corresponding observed ground truth rating.

Table 2.1: Ratings and predictions for products of a fictitious retailer on a 1 to 5 scale. In this example, products rated by 4 or higher are classified as relevant.

Product	Rating (r)	Prediction ($\hat{r}$)	Relevance (rel)	Rank	Percentile Rank
Trousers	5	4	1	1	0.25
T-Shirt	3	4	0	2	0.5
Jacket	4	3	1	3	0.75
Pullover	1	1	0	4	1.0

The **RMSE** is defined as follows:

$$\text{RMSE} = \sqrt{\frac{\sum_{r_{u,i} \in R} (r_{u,i} - \hat{r}_{u,i})^2}{|R|}} \tag{2.20}$$

The RMSE determines the overall prediction error by considering the squared distance between the predicted rating and its ground truth. At the end, the square root is taken.

The **MAE** is defined as:

$$\text{MAE} = \frac{\sum_{r_{u,i} \in R} |r_{u,i} - \hat{r}_{u,i}|}{|R|} \tag{2.21}$$

In contrast to the RMSE, the MAE considers the magnitude of the predicted rating and its ground truth.

Considering the ratings and predictions in Table 2.1 the RMSE and MAE are determined in Equation 2.22 and 2.23, respectively.

$$\text{RMSE} = \sqrt{\frac{(5-4)^2 + (3-4)^2 + (4-3)^2 + (1-1)^2}{4}} \approx 0.87 \tag{2.22}$$

$$\text{MAE} = \frac{|5-4| + |3-4| + |4-3| + |1-1|}{4} = 0.75 \tag{2.23}$$

Classification Accuracy

In contrast to the prediction accuracy, the classification accuracy aims to predict whether a specific item is of relevance or utility for a user. Generally, an item might be classified as relevant since it received a high rating or was purchased by the user. Hence, the evaluation is based on a list of n items provided to a user, so-called top-N recommendations. In this scenario, a recommender system's task is to generate a list of n recommendations containing as many relevant items as possible. In the following, n_{rs} defines the number of relevant and selected items, n_s the number of selected items, and n_r the number of relevant items as applied in [HKTR04].

Precision is defined as follows:

$$\text{precision} = \frac{n_{rs}}{n_s} \tag{2.24}$$

The precision determines the ratio of the number of relevant and selected items n_{rs} to the selected ones n_s. In other words, it measures how many of the recommended items are considered as relevant [HKTR04, p. 23].

Recall is defined as:

$$\text{recall} = \frac{n_{rs}}{n_r} \tag{2.25}$$

In contrast to precision, recall describes the ratio of the number of relevant and selected items n_{rs} to the relevant ones n_r, i.e., how likely is it to recommend a relevant item [HKTR04, p. 23].

The harmonic mean of precision and recall is termed as **F1**. It is an important metric that shows a balance between these two metrics and is expressed as:

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{2.26}$$

Considering a recommendation list containing the products *Trousers*, *T-Shirt*, *Jacket*, and *Pullover*, as shown in Table 2.1, the number of relevant and total items n_{rs} is 2 and the number of selected items n_s is 4. This leads to a precision of $\frac{2}{4} = 0.5$. On the other hand, the number of relevant items n_r is 2, resulting in a recall of $\frac{2}{2} = 1.0$. Consequently, applying Equation 2.26 leads to the following F1 score:

$$\text{F1} = 2 \cdot \frac{0.5 \cdot 1.0}{0.5 + 1.0} = \frac{1.0}{1.5} = \frac{2}{3}.$$

Ranking Accuracy

The ranking accuracy goes beyond the simple classification of items into relevant and irrelevant. It extends the evaluation by not only considering whether a particular item is part of the top-N recommendation but also its position in the recommendation list. The idea is to reward recommender systems, generating a ranked list with relevant items at the top and penalizing the ones containing irrelevant items. In the following, some of the most established metrics to measure ranking accuracy are discussed.

Discounted Cumulative Gain (DCG) and its normalized version are defined as follows:

$$\text{DCG(L,u)} = \sum_{i=1}^{N} \frac{rel_{u,i}}{\log_2(i+1)} \tag{2.27}$$

$$\text{nDCG(L,u)} = \frac{\text{DCG}(L,u)}{\text{DCG}(L_{\text{ideal}},u)} \tag{2.28}$$

In Equation 2.27, the parameters specify a ranked list of items L for a user u. The size of the ranked list L is defined by N. Furthermore, $rel_{u,i}$ indicates whether the i-th item is relevant for user u and is weighted by its position in the ranked list L. Finally, Equation 2.28

normalizes the result by dividing by the DCG value of an ideal ranked list L_{ideal}.

The following example, based on the product ranking in Table 2.1, illustrates the application of normalized Discounted Cumulative Gain (nDCG). Here, the ranked list L is defined as:

$$L = [\text{Trousers}, \text{T-Shirt}, \text{Jacket}, \text{Pullover}].$$

On the other hand, the ideal ranked list L_{ideal} is represented as:

$$L_{ideal} = [\text{Trousers}, \text{Jacket}, \text{T-Shirt}, \text{Pullover}].$$

In this case, the DCG applied to L leads to:

$$\text{DCG}(L, u) = \frac{1}{\log_2(2)} + \frac{0}{\log_2(3)} + \frac{1}{\log_2(4)} + \frac{0}{\log_2(5)} = \frac{1}{1} + \frac{1}{2} = 1.5.$$

And the DCG for the ideal ranked list L_{ideal} leads to:

$$\text{DCG}(L_{\text{ideal}}, u) = \frac{1}{\log_2(2)} + \frac{1}{\log_2(3)} + \frac{0}{\log_2(4)} + \frac{0}{\log_2(5)} = \frac{1}{1} + \frac{1}{1.585} = 1.63.$$

Consequently, applying Equation 2.28, the final nDCG value is $\frac{1.5}{1.63} = 0.92$.

Another popular metric is the **Mean Percentile Rank (MPR)** as defined in Equation 2.29. It measures an item's percentile ranking in a recommendation list, whereas a rank value of 0% indicates high and 100% low preference. In other words, the lower the MPR, the better the generated recommendation list.

$$\text{MPR} = \frac{\sum_{u,i} rel_{u,i} \cdot \text{rank}_{u,i}}{\sum_{u,i} rel_{u,i}} \tag{2.29}$$

Here, $rel_{u,i}$ defines the relevance of item i for user u and $rank_{u,i}$ the according percentile ranking in the recommendation list. Applying the metrics to the example in Table 2.1 leads to:

$$\text{MPR} = \frac{(1 \cdot 0.25) + (0 \cdot 0.5) + (1 \cdot 0.75) + (0 \cdot 1.0)}{2} = \frac{1}{2} = 0.5.$$

Considering a list of 4 items, the percentile is divided into 0.25, 0.5, 0.75 and 1.0. Then, each item is weighted by its relevance and ranking percentile. Finally, the result is divided by the total number of relevant items in the list. In this case, on average, a relevant item appears within the first 50% of the recommendation list. It must be noted that the MPR is rather intended to be applied to the complete test data set than on a per user basis.

Besides, **Mean Reciprocal Rank (MRR)** is widely used and defined as follows:

$$\text{MRR} = \frac{\sum_{u \in U} \frac{1}{\text{rank}_u}}{|U|} \tag{2.30}$$

Here, $rank_u$ is the position of the first relevant item in the recommendation list of user u and $\frac{1}{rank_u}$ the corresponding reciprocal. For the mean value, the reciprocal ranking is determined for all users U and divided by its size $|U|$. Considering Table 2.1 again, *Trousers* is relevant and recommended first, such that $rank_u = 1$. This leads to a reciprocal ranking of $\frac{1}{1} = 1$.

2.5.2 Non-Accuracy Metrics

The following part compiles and explains non-accuracy metrics from [RRS11, p. 281-293], [RMWZ14, p. 246-267] and [KB16].

Coverage
The coverage of the recommendations can be considered from a user and item perspective. User coverage measures for how many users the recommender system could generate recommendations [JZFF10, p. 183]. On the other hand, item or catalog coverage aims to measure how many of the available items are part of recommendations. The catalog coverage can be further examined by the Gini Index and the Shannon Entropy, which additionally consider how the items are distributed [RRS11, p. 282].

Diversity
Diversity aims to measure whether the provided items within a list of recommendations are rather similar or diverse. The idea of this measure is to avoid showing too many similar items to the user and instead generate a recommendation list covering more items of the catalog [JZFF10, p. 183]. In doing so, a recommender system avoids that a user stucks in a filter bubble. A commonly used metric in this context is the Intra-List Similarity (ILS).

Confidence
As discussed in [RRS11, p. 283], [STS$^+$12, p. 283], confidence tries to measure how sure the recommender system is that the generated recommendation will be trusted by a user.

Novelty
The idea of novelty is to provide recommendations to users, they might not be aware of. In other words, items that are unknown to the user. One approach in this regard is to exclude already known and popular items from the recommendation list of a user, assuming that these items are more likely to be known by him or her [RRS11, p. 285-286].

Serendipity
According to [RRS11, p. 287-287], serendipity aims to recommend surprising, unexpected, and novel but still relevant items to a user. One approach might be to add some random items into the recommendation list [JZFF10, p. 76]. A survey on the topic is provided by Kotkov et al. in [KWV16].

Adaptivity
In [RRS11, p. 292], adaptivity is defined as the capability of a recommender system to adapt due to fast changes in the catalog. Besides, adaptivity also includes a recommender system to adapt to changes in user preferences. A possible way to measure the adaptivity is to compare the recommendations for a user before and after including new information [RRS11, p. 292].

Robustness
Despite adaptivity, it is also important for a recommender system to be robust against attacks that try to impact the recommendation results [RRS11, p. 290]. Such an attack might involve fake accounts giving high rates to poor items so that these items appear as recommendations for other users.

Learning Rate
According to [STS$^+$12], [RMWZ14, p. 283], the learning rate measures the accuracy of a recommender system over time. In this case, the underlying assumption is that the recommender system generates better recommendations when more data is available. As a consequence, problems such as the cold start problem can be alleviated [RMWZ14, p. 283].

Calibration
Calibration aims to evaluate whether the generated recommendations preserve the proportions of past user interests. In other words, when a user shows a balanced interest in the topic of recommender systems and benchmarking, the recommendations should reflect this ratio, as discussed by Steck in [Ste18].

Click-through Rate and Conversion Rate
The Click-through Rate (CTR) measures how many presented recommendations are actually clicked by a user. Consequently, it provides useful information about the efficiency and acceptance of the applied recommender system. The Conversion Rate (CR) measures, for instance, how many of the recommendations lead to actual purchases. In this regard, both represent important business metrics, which are mostly applied in the online evaluation of recommender systems, as discussed by Antonino Freno for Zalando in [Fre17] or Wang et al. for Alibaba in [WBCR17].

Competitive Ratio
Traditional recommender system algorithms compute recommendations offline and periodically, e.g., on an hourly or a daily basis, considering the complete input and data. On the other hand, streaming recommender systems aim to incorporate new information continuously to keep their corresponding models updated. In this sense, it might be of interest to compare streaming recommender systems with traditional recommender systems, whereas the latter serves as a baseline. In computer science, this relates to determining the competitive ratio among the two approaches. In other words, the trade-off between scalability and accuracy.

Others
Other mentioned non-accuracy metrics are utility [RRS11, p. 289-290], risk [RRS11, p. 290] and privacy [RRS11, p. 291]. They are mentioned for the sake of completeness but are not clearly defined in the literature and hard to evaluate. For instance, the utility of a recommender system can be evaluated from a business or user perspective. Risk considers to what extent the provided recommendations may lead to harmful user decisions, e.g., gambling on the stock market. Privacy aims to avoid revealing private information about user interests in the recommendation process.

2.5.3 Performance Metrics

Scalability

Scalability can be considered from a system and algorithmic view. One option to increase a system's performance is to add additional hardware resources, termed vertical scaling. The other option, horizontal scaling, aims to improve performance by distributing a system as a cluster, e.g., Database Management Systems (DBMSs). From an algorithmic view, scalability describes how well it performs to a growing input. More precisely, the "computational complexity of an algorithm in terms of time or space" [RRS11, p. 293]. Since the number of users and items is continuously increasing, it is critical to evaluate these aspects [RRS11, p. 293]. For instance, whether an increase of the user has a linear or exponential effect on the runtime. In computer science, these aspects are termed polynomial and non-polynomial complexities [Weg03]. In the industry, the most popular metrics are Central Processing Unit (CPU) runtime and memory consumption.

Reactivity

Another important aspect is to consider how fast a recommender system is able to generate and provide recommendations to a user [RRS11, p. 293], [STS$^+$12]. At this point, the throughput and response times of a recommender system are significant metrics. In this case, throughput measures the number of recommendations that can be generated, for instance, per second. The response time measures the time a user has to wait to retrieve recommendations, for instance, after its login.

2.6 Evaluating Recommender Systems

Besides the *"what"* to measure, defined by the metrics, the *"how"* to measure is discussed next. In general, the goal of an evaluation is to assess, for instance, algorithms or systems, considering their performance [BCT18, p. 295-328]. In the case of recommender systems, the evaluation mostly focuses on the accuracy of the generated recommendations. In the beginning, especially, the accuracy of predicting ratings was investigated [BCT18, p. 295]. According to [BCT18, p. 296], focussing on this aspect might even lead to useless recommendations for a user. Nowadays, the focus has changed to evaluate the order of the generated recommendations, also referred to as ranking, within a list instead, since it is considered more beneficial for a user [BCT18, p. 296]. Besides accuracy, other aspects, for instance, coverage, diversity, novelty, or serendipity of recommendations, are receiving more attention [BCT18, p. 296]. Also, aspects considering the deployment and maintenance, as well as the runtime and memory consumption, might be part of the evaluation [BCT18, p. 303],[JZFF10, p. 167]. However, for an evaluation, it is important to define clearly what is going to be evaluated by specifying an objective function [BCT18, p. 307]. Beyond that, a systematic process and its documentation are an integral part of each recommender system evaluation to ensure the reproducibility of the achieved results [JZFF10, p. 168]. In the literature, recommender systems are commonly evaluated by offline and online experiments, referred to as offline and online evaluation [JZFF10, BCT18, RRS11].

2.6.1 Offline Evaluation

In offline evaluation, a recommender system is evaluated on static data, which represents the historical preferences of users for items. In doing so, offline evaluation assumes that the user behavior of the historical data is similar to the user behavior "when the recommender system is deployed" [BCT18, p. 261]. In this regard, one goal of an offline evaluation might be to narrow down the number of possible recommender system approaches [RRS11, p. 261]. Preferences are generated by users in explicit or implicit form as ratings or purchases, respectively. In this context, the data might not represent real user preferences but rather synthetically generated ones [JZFF10, p. 169]. Since the available data is static, it is possible to examine many algorithms with different parameter settings [BCT18, p. 297]. A common approach, adapted from ML, is to split the available data into training and test data. The chosen splitting method directly influences the results of the applied algorithms [BCT18, p. 298]. Therefore, in the following, some widely used splitting methods are discussed.

N-fold cross-validation
This method splits the data into N disjoint partitions, where $N-1$ partitions are used for training and the remaining partition for testing. Consequently, during the evaluation, each partition is used $N-1$ times for training and exactly once for testing [JZFF10, p. 177].

Leave one out
This method considers each preference of a user as a partition. Consequently, during the evaluation, all available user preferences are used for training except one, which is used for testing [JZFF10, p. 178].

All but N
This method splits the data in such a way that the generated training data set contains exactly N item preferences for each user [JZFF10, p. 178]. A critique of this method is that it does not reflect the real world since it is not the case that all users have N item interactions [RRS11, p. 262]. In this regard, such a splitting method shows under which conditions an algorithm performs best [RRS11, p.263] since it specifically evaluates how many preferences must be at least given for each user to achieve high accuracy.

Given N
This method splits the data in such a way that the generated testing data set contains exactly N item preferences for each user [JZFF10, p. 178]. Similar to all but N, given N might not represent the real world and therefore rather evaluates the best-case scenario for a recommender system.

Time-based
This method is only applicable when the data includes a timestamp for each user preference. Then, the data is split according to a specified fixed point in time t, meaning that the training data set contains all preferences appearing before t, whereas the testing data contains those appearing after t [BCT18, p. 298]. Due to the fact that recommender systems aim to predict future interactions based on historical data, time-based splitting is considered to be a good representative of real-world scenarios [BCT18, p. 298].

P-cores
This method is used to reduce the sparsity within a given data set. To do so, only user and items interactions remain in the data set, which have at least p interactions [BCT18, p. 299]. However, such a preprocessing of the data set might introduce bias and should, therefore, be applied carefully [RRS11, p. 262].

Mask-based
The idea of mask-based splitting is to generate train data by randomly masking a certain percentage p of its entries [51]. In doing so, these entries are hidden from the recommender system during its training. In contrast to the other methods, the test data is a copy of the initial data, such that the train and test data are not disjoint. Finally, during the evaluation, for all users having at least one masked entry in the train data, a recommendation list is generated and compared to the test data. In this thesis, the evaluation also only considers users who were part of the training data.

2.6.2 Online Evaluation

In contrast to offline evaluation, the results obtained by an online evaluation are regarded to be more trustworthy since they are based on a deployed recommender system retrieving direct feedback from the users [BCT18, p. 297]. For instance, in the offline evaluation, a recommendation for an item might be made that is not part of the test data of a specific user. Nevertheless, the item might be of interest to him or her, but the user was simply not aware of it and, therefore, not able to indicate a preference. In this case, the recommender system is penalized for recommending it since the item is classified as irrelevant [JZFF10, p. 171]. Such recommendations can be made and directly evaluated during an online evaluation since a user either ignores the recommendation or utilizes it.

Another advantage of online evaluation is its ability to check the usability of the graphical representation of the recommendations, for instance, how many items to recommend or where to place the recommendations [BCT18, p. 506-507]. There exist two commonly applied online evaluation approaches that will be discussed in the following.

A/B Testing
In A/B testing, the population of users is separated randomly into different groups. In this regard, a user does not know to which group he or she belongs, nor being part of an experiment at all [BCT18, p. 305-306]. Each of these groups uses a different recommender system implementation. Consequently, the results of the different recommender system implementations are examined and conclusions are made. For instance, a company might be interested in whether a new recommender system performs better than the current one. To reduce the uncertainty about the quality of the new approach, the company may decide first to deploy it to a small group of users and gain some insights before providing it to all of its users. In doing so, a company lowers the risk of losing its reputation due to a bad user experience [BCT18, p. 406].

User Studies

According to [BCT18, p. 306], user studies represent a qualitative evaluation of a recommender system. This is based on the fact that the feedback from the users is collected by, for instance, questionnaires, interviews, or during the execution of given tasks. In this regard, a company is able to create concrete and goal-oriented tests in a controllable environment. Thus, in user studies, the users are aware of be being part of a study. Since participation in user studies involves additional effort for the users, incentives are given [BCT18, p. 306]. As a consequence, user studies are more costly to conduct and evaluate [RRS11, p. 264]. Furthermore, the number of participants in user studies is much smaller compared to A/B testing. This leads additionally to the question of how to select representative participants for the study [RRS11, p. 265]. Besides, the results might be biased since the users know to be part of a study and therefore behave differently [RRS11, p. 265].

2.7 Evaluation Libraries and Frameworks

Since recommender systems represent an active research field, multiple libraries exist, which provide implementations of the most established algorithms. In addition, most of these libraries contain built-in metrics to measure the accuracy of the algorithms. Besides, there exist specific frameworks focusing on the evaluation of recommender systems. Therefore, this section aims to give an overview of the most popular libraries, frameworks, and services.

2.7.1 Libraries

LensKit[18]

LensKit is a Python and Java-based library that supports the evaluation of various recommender system algorithms. Therefore, the library comprises not only implementations of algorithms but also data sets, splitting methods, and common metrics. Currently, the library supports CF algorithms such as item-based and user-based kNN, as shown in Listing 2.1. Additionally, the MF implementation of Implicit and FunkSVD are available. The built-in metrics for evaluation are classified into prediction, classification, ranked list, and utility metrics. The metrics are RMSE, MAE, precision, recall, RR and nDCG, as discussed in Section 2.5. Furthermore, the library provides interfaces to implement own algorithms. In [Eks18] additional information about the library is available.

```python
from lenskit.algorithms import item_knn as knn

item_item_cf = knn.ItemItem(nnbrs=20)
item_item_cf.fit(train_data)
recommendations = item_item_cf.recommend(user)
```

Listing 2.1: Example of applying item-based CF with LensKit. Here, $nnbrs$ defines the number of neighbors.

[18]https://lenskit.org

Implicit[19]

The Implicit library provides implementations of the following algorithms: Alternating Least Squares (ALS), Bayesian Personalized Ranking (BPR), and Logistic Matrix Factorization (LMF). The implementations of the algorithms are heavily optimized by using Cython[20] and OpenMP[21] to parallelize the model generation. As the name implies, the library focuses on recommender systems based on implicit user feedback. An example code is shown in Listing 2.2. Implicit provides support for the evaluation of the recommender system by a simple splitting method and metrics. These include precision, MAP, nDCG, and AUC. However, LensKit also includes the algorithms of Implicit so that it is easier to evaluate the results in that way.

```
import implicit

als = implicit.als.AlternatingLeastSquares(factors=100,iterations=20)
als.fit(train_data)
recommendations = als.recommend(user)
```

Listing 2.2: Example of applying ALS with Implicit. Here, $factors$ denotes the dimension of the vectors and $iterations$ the number of iterations.

Surprise[22]

The Surprise library is mainly specialized for evaluating recommender systems based on explicit user feedback. Currently, it supports multiple algorithms such as kNN based ones, MF approaches such as SVD, SVD++, and Non-negative Matrix Factorization (NMF). Besides, Slope one and Co-Clustering (CoC) are part of the library. Furthermore, Surprise comprises a data set, a splitting, a similarity, and an accuracy module, thereby covering most aspects of an evaluation. The available accuracy metrics are RMSE, MSE, MAE and FCP. Listing 2.3 shows how to implement SVD with the Surprise library.

```
from surprise import SVD

svd = SVD(n_factors=100,n_epochs=20)
svd.fit(train_data)
prediction = svd.predict(user, item)
```

Listing 2.3: Example of applying SVD with Surprise. Here, $n\_factors$ defines the dimension of the vectors and n_epochs the number of iterations.

FluRS[23]

FluRS is a Python-based library for online recommendations [43]. Therefore, the library provides implementations of algorithms, which adapt their models incrementally, such as incremental CF, incremental MF, incremental MF with Bayesian Personalized Ranking (BPR) optimization, or incremental Factorization Machines [43]. For evaluation purposes, the library

[19]https://implicit.readthedocs.io/en/latest/index.html
[20]https://cython.org
[21]https://www.openmp.org
[22]http://surpriselib.com
[23]https://takuti.github.io/flurs/

contains the following metrics: precision, recall, AUC, RR, MPR, and nDCG. According to the developers [43], the library is based on user, item, and event entities. An event is described by its value, e.g., purchase and its context, e.g., time or location [43]. In order to provide flexibility, algorithms and recommenders are implemented separately.

MyMediaLite[24]

MyMediaLite is a .NET-based library developed at the University of Hildesheim. The library targets to evaluate CF algorithms, particularly rating prediction in case of explicit feedback and item prediction in case of implicit feedback. Part of the library are implementations of nearest neighbor, e.g., user-based and item-based, as well as MF approaches, e.g., SVD [8, 7]. For evaluation purposes, it provides metrics such as RMSE, MAE, AUC, precision, MAP, and nDCG. Further information about the library is available in the following paper [GRFST11].

2.7.2 Frameworks

OpenRec[25]

In [YBG$^+$18], the authors introduce the Python framework OpenRec. The goals of the framework are to provide extensibility and adaptability by a modular architecture of a recommender system. At this point, a recommender system represents a composition of modules for extraction, fusion, and interaction. By their defined interfaces, it is possible to connect the modules easily. An extraction module aims to determine "representations for a data trace from users, items, or contexts" [YBG$^+$18, p. 667]. In some cases, a recommender system may incorporate various data sources, which leads to multiple implementations of extraction modules. At this point, a fusion module takes care of combining the extraction modules. In addition, an interaction module uses the input of a fusion or extraction module for training and testing. Besides, the framework includes utility functions for splitting data and evaluating results. In contrast to other libraries, OpenRec leverages modern hardware capabilities such as GPUs through TensorFlow[26] to speed up the computation.

RiVal[27]

RiVal is a Java-based framework focusing on the evaluation of recommender systems [SB14a, SB14b]. As discussed in [SB14a], recommender system libraries provide functionality for splitting data, applying algorithms, and measuring results, but even though the parameters are chosen similarly among the libraries, the results vary considerably. Therefore, to achieve a fair comparison, RiVal takes care of data splitting, candidate generation, and performance measurement [SB14a], only leaving the item recommendation part to the libraries.

Idomaar[28]

Idomaar [SLK$^+$16] aims to evaluate a recommender system in a realistic environment, reflecting industry requirements. In this regard, besides accuracy, e.g., RMSE, recall, and precision,

[24]http://www.mymedialite.net
[25]https://openrec.ai
[26]https://www.tensorflow.org
[27]http://rival.recommenders.net
[28]https://github.com/crowdrec/idomaar

the technical aspects, for instance, the throughput and the response time of recommender systems, are part of the evaluation. In contrast to other frameworks that evaluate recommender systems on static data, Idomaar considers data as a continuous stream of new information about users, items, and their interactions. In doing so, Idomaar tries to bridge the gap between offline and online evaluation. The proposed generic architecture comprises a data container, an evaluator, an orchestrator, and a computing environment [SLK$^+$16]. The authors provide an implementation of the architecture in the form of a virtual machine. Here, Vagrant[29] is used to setup the internal components, such as Apache Flume[30] to read and Apache Kafka[31] to process the data. The evaluator takes care of splitting the data and evaluating the results. As the name implies, the orchestrator manages the lifetime of the computing engine as well as the execution of the evaluator [SLK$^+$16]. A streaming scenario can be realized by ingesting the data directly into Apache Kafka.

AlpenGlow[32]
In [FPK$^+$17] the C++-based evaluation framework AlpenGlow is introduced, which additionally provides a Python API. The framework aims to support a fast implementation of recommender systems by providing so-called environments to conduct offline and online experiments. In this regard, the authors emphasize their goal to evaluate top-N recommender systems in dynamic application scenarios. To do this, the framework provides online experiments for realistic recommender systems, which update their models incrementally whenever new information about a user is available. Nevertheless, offline experiments are available, covering the traditional evaluation of recommender systems based on historical batch data. According to [FPK$^+$17], the framework supports, for instance, batch and online MF approaches as well as SVD++ and their variations. In addition, the framework allows combining batch and online approaches based on timely conditions. This means that a recommender system might be trained weekly by a complete batch of data and afterward updated incrementally on a daily basis. Currently, only MSE, DCG, precision, and recall are supported during an experiment.

sRec
sRec is a framework to generate recommendations based on streaming data [FPK$^+$17]. In this regard, the authors consider three challenges streaming recommender systems have to tackle: real-time updating of models, an unknown number of users and items, and concept shifts in the popularity of items or user preferences. The general architecture of sRec comprises one component to estimate appropriate parameter settings based on historical data, whereas the other component updates the models and generates recommendations based on streaming data [FPK$^+$17, p. 383]. The framework provides a self-developed recommender system approach, which is compared against Probabilistic Matrix Factorization (PMF), Multi-Dimensional Collaborative Recommendation (MDCR), Time-SVD++, and Gaussian Process Factorization Machines (GPFM). The algorithms were applied to the following data sets: MovieLens Latest Small, MovieLens-10M, and Netflix and compared by their resulting

[29]`https://www.vagrantup.com`
[30]`https://flume.apache.org`
[31]`https://kafka.apache.org`
[32]`https://alpenglow.readthedocs.io/en/latest`

RMSE.

StreamingRec[33]
The open-source framework StreamingRec focuses on the application domain of news [JJK18]. One challenge in this specific application domain is to handle an endless stream of new and popular items of the last couple of minutes instead of hours and weeks as in other domains. Consequently, a recommender system has to be able to update its models continuously to provide appropriate recommendations to its users. This leads to the other challenge of finding recommender system algorithms that are capable of adapting quickly. As a solution, the authors propose to use session-based and context-aware recommender system approaches. StreamingRec is based on a "replay-based evaluation protocol" [JJK18, p. 1], which aims to address these challenges. The framework comes with some algorithmic implementations such as most popular, recently popular, recently clicked, recently published. Besides, an item-based CF approach, as well as a session-based kNN (SkNN) and its extension v-SkNN, are part of StreamingRec [JJK18, p. 3]. Finally, two rule-based and two CB algorithms are implemented. During the evaluation, the data sets Outbrain and Plista were used. Furthermore, a time-based approach was applied with 70% for training and 30% for testing. As baseline algorithms, the authors used Bayesian Personalized Ranking (BPR) and GRU4Rec. The results of the evaluation are reported by accuracy, for instance, F1 and MRR. Additionally, diversity, as well as the computation time, are considered in the results.

StreamRec
In [CLEM11], the authors present a CF recommender system for streaming data, called StreamRec. More specifically, StreamRec is implemented by leveraging a stream processing system, which enables high scalability. In their implementation, the authors use Microsoft StreamInsight for processing data. In doing so, the recommender system only utilizes "native incremental streaming operators" [CLEM11, p. 1] in a well-defined query plan. The query plan is able to build a model as well as generate recommendations, depending on the input event type. Here, two stream event types are distinguished: update and recommend events. An update event represents interactions of users with items, whereas recommend events represent requests for new recommendations. In this context, an event is represented as a tuple of the following form: $(Timestamp, StreamId, UserId, ItemId, Rating)$. The authors demonstrate the usability of the recommender by providing two exemplified implementations: MSRFlix and MSRNews. MSRFlix uses the MovieLens data set and MSRNews the Digg data set.

FAiR
In [CSS$^+$18], the authors introduce FAiR - A Framework for Analysis in Recommender Systems. The source code is publicly available on GitHub[34]. FAiR aims to support researchers in the selection and implementation of evaluation metrics as well as in the comparison of the results. The authors classify metrics into effectiveness-based metrics, complementary dimensions of quality, and domain profiling [CSS$^+$18, p. 2]. Effectiveness-based metrics are, for instance, precision or recall [CSS$^+$18, p. 4-5]. Novelty, serendipity, and diversity are considered as

[33]`https://github.com/mjugo/StreamingRec`
[34]`https://github.com/dcomp-labPi/FAiR`

complementary dimensions [CSS$^+$18, p. 6]. On the other hand, domain profiling represents statistical metrics such as item popularity or the average rating of users and items [CSS$^+$18, p. 7]. To use FAiR in a convenient way, the authors provide a graphical user interface to import the input files, which includes a train, a test, a feature, and a recommendation file, as well as to configure metrics.

2.7.3 Recommender System Services

Beyond that, some projects provide complete recommender system services to simplify their deployment, execution, and maintenance.

PredictionIO[35]
PredictionIO is a server suited for developing ML tasks such as recommendation, classification, and clustering. The architecture of PredictionIO comprises an event server and one or multiple so-called engines. At this point, an event server is responsible for collecting and processing incoming data for the engines. An engine is implemented in a DASE architecture, which stands for Data Preparator, Algorithm, Serving, and Evaluation Metrics. This architecture enables to create own engines by implementing the defined interfaces of the corresponding component as needed. The default template configuration uses HBase[36] as an event server, Spark to train ML models, and Hadoop Distributed File System (HDFS) to store the data sets and the model. Besides, Elasticsearch is used to store general metadata. To deploy a server effortlessly, PredicitionIO provides a Docker container and supports Docker Compose to combine the technologies. Furthermore, PredictionIO provides a template gallery with recommender systems.

Oryx 2[37]
Similar to PredictionIO, Oryx represents an open-source[38] ML server implemented by the lambda architecture. In this sense, Oryx 2 is especially suited for real-time ML applications, for instance, CF, clustering, and classification [14]. Since the architecture is based on the lambda architecture, it comprises a batch, a speed, and a serving layer. Oryx 2 stores incoming events in a HDFS[39]. The incoming events are collected via the serving layer, which is implemented by an Apache Tomcat[40] web server with an exposed HTTP REST API for communication. In the process, Apache Kafka[41] is used to transfer the events to the batch and streaming layer. At this point, the batch layer takes care of storing the events in HDFS. Additionally, the batch layer periodically, mostly hours or even days, generates ML models by data retrieved from HDFS. For this purpose, Oryx 2 uses Apache Spark[42]. Afterward, the models are provided to the streaming layer, which incorporates the events of the last couple of seconds into the

[35] `https://predictionio.apache.org`
[36] `https://hbase.apache.org`
[37] `http://oryx.io/`
[38] `https://github.com/OryxProject/oryx`
[39] `https://hadoop.apache.org`
[40] `http://tomcat.apache.org`
[41] `https://kafka.apache.org`
[42] `https://spark.apache.org`

model. Finally, the updated model is transferred to the serving layer by Apache Kafka and used to respond to new queries. The performance of Oryx 2 is evaluated considering its memory consumption (heap size), latency (milliseconds), and throughput (queries per second) by various combinations of feature, item, and user sizes.

Froomle[43]

Froomle provides a service to personalize a user's experience during his or her interaction with a website. This includes, for instance, presenting personalized recommendations in real-time, personalized search results as well as generating personalized email content. Furthermore, personalized communication with a chatbot is possible, considering the current user's preferences.

The goal of Froomle is to alleviate the integration of a recommender system for their customers by taking care of the internal implementation details, such as scalability or algorithm selection. Their customers' application domains range from news to online retailing, which leads to different requirements and, therefore, customer-specific adaptations. For instance, in the news domain, new articles appear frequently and therefore lead to other requirements as in online retailing, where the number of products is more stable. Consequently, the underlying system architecture of Froomle has to be implemented in a flexible and adaptable manner.

To achieve this, the architecture of Froomle is hosted on the Google Cloud Platform[44]. The architecture provides multiple REST APIs covering different aspects of the system. According to the official documentation [20], Froomle provides APIs for configuration, search, recommendation, batch recommendation, events, and metrics. Besides, customers can upload meta item data via Secure File Transfer Protocol (SFTP), for instance, to keep track of the item availability in the recommendation process. This kind of data is stored within a cloud storage system separately from the other data and is mostly uploaded once a day, but other intervals are also possible. As the name implies, the events API collects events generated by users during their interaction with a customer's website. Events are distinguished by different types, such as tracking events, e.g., visiting a page, integration events, e.g., clicking on a recommendation, or retail events, e.g., purchasing an item. In the process, the collected events are processed by an asynchronous publish-subscribe message queue and forwarded to an event and history writer. The events writer stores all incoming events in the Parquet[45] file format, whereas the history writer stores events such as page views, purchases or impressions in a Scylla[46] database. A model recomputation component leverages the events processed by the events writer in combination with the available item metadata and stores the recommendation models. Finally, the recommendation service component takes care of loading the recommendation models into memory and generating recommendations based on the events stored in the Scylla database. Moreover, it applies additional filter operations on the recommendation results, for instance, to either present similar or complementary items and check whether an item is currently available.

In Froomle a recommender system is highly configurable through multiple configuration

[43]`https://www.froomle.ai`
[44]`https://cloud.google.com`
[45]`https://parquet.apache.org`
[46]`https://www.scylladb.com`

files, which compose an environment for a customer. One part of the configuration, for instance, defines how to transfer incoming events to the internal data representation of Froomle. In addition, it is possible to define which events are considered by the applied algorithms. At this point, Froomle basically utilizes a popularity and an item-based CF approach. The parameters for the popularity-based approach are, for instance, the time interval popular items are generated for and the event types to take into account. On the other hand, the configuration of item-based CF involves the number of nearest neighbors, the minimum number of user-item interactions, and the considered event types. For both approaches, Froomle generates the recommendations in real-time but additionally provides a batch mode to retrieve recommendations for multiple users. In this regard, the location to store the recommendation models can be specified. Moreover, Froomle is able to apply different recommender system approaches depending on the current user context or device. For instance, depending on whether a user's current context is a home page, an overview page, or a product detail page. Furthermore, the internal representation of the provided metadata of a customer is configurable since it might differ among different application domains.

In order to evaluate different recommender system approaches online, it is possible to apply different recommender systems and thereby other recommendations to users depending on their assigned group.

2.8 Industrial Recommender System Implementations

In the following, selected industrial recommender system implementations of Netflix, Mendeley, and Zalando are explained. The selection represents one implementation of the application domains e-resource, e-library, and e-commerce as introduced in Section 2.2. The implementations provide insights into how recommender systems are deployed in an industrial environment and emphasis that algorithms are just one part of a recommender system, which compose the overall architecture. Furthermore, it shows which technologies are applied to run the recommender systems in real-world scenarios.

2.8.1 Netflix Recommender System

Netflix was founded in 1997 and initially provided a service to rent DVDs via mail. In the following years, Netflix moved into a video streaming provider with approximately 182 million users across the world [11]. Nowadays, Netflix additionally evolved into a movie and series production company. Due to its business model of providing its customers with flexible monthly subscriptions to access their content and the growing number of competitors in the field of video streaming, Netflix aims to create high user engagement and retention. One major cornerstone in this regard is to provide a customer with the content he or she is most interested in. In the past, customers went into their video rental store of trust and may be asked the owner for suggestions. Nowadays, the situation changed, customers are not willing to spend much time searching for content on their own but rather receive interesting content automatically. What sounds like a simple task to do, indeed, is challenging because of the following reasons. First, the amount of possible movies and series to choose from is extended continuously and overwhelming. And second, the number of customers using the service is increasing from year to year. In this regard, Netflix's success is highly dependent on an accurate and well-engineered recommender system.

In 2013, Netflix, for the first time, granted some insights into the involved technologies of their recommender system. The description is available in a blog entry by Xavier Amatriain and Justin Basilico [53]. Their architecture comprises three layers denoted as offline, nearline, and online, as illustrated in Figure 2.4. In the online layer of the recommender system, the customers interact with the Netflix service by using smartphones, tablets, TVs, or the web browser on his or her computer. During an interaction with the service, Netflix collects all information about his or her behavior. This involves information about what movies and series the customer has played, rated, and browsed.

Additionally, Netflix stores what information was presented to the customer during its interaction, which they refer to as impressions internally. For instance, which genres have been presented to the customer, what movies were part of each genre, and what was the ordering of the movies within each genre. Furthermore, information such as the used device and the time of playing are of interest to Netflix. All this information is collected via the clients, transferred to Netflix and stored.

In its first implementation, Netflix used Apache Chukwa[47] to process the massive amount of event data generated by the customers. In the next step, the event data is distributed to be stored in its raw format in HDFS for heavy and expensive ML algorithms running regularly in the offline layer. To query the appropriate data to train models via ML approaches Apache Hive[48] and Apache Pig[49] are used. Because of the runtime of the queries, Netflix uses a system called Netflix.Hermes to propagate the results to ML algorithms, which use the data to build models out of it. In addition, offline data and external data, such as box office performance and reviews of movie critics, are processed. The trained models are used within the offline layer to precompute intermediate recommendation results but are also available in the online layer.

Furthermore, the user event data is stored into different database systems, such as Apache Cassandra[50], Oracle MySQL[51] and EVCache[52]. As explained in [53], the reason for that is related to balance the advantages and disadvantages of the different technologies.

According to their technical blog post in 2016 [26], Netflix stepwise adapted its architecture. The reason behind that was to reduce the latency for data analytic tasks. With the advent of new streaming technologies, such as Apache Kafka[53], the overall latency could be reduced from approximately 10 minutes to sub-minutes. In the first step, Netflix kept the initial implementation and just extended its current infrastructure by Apache Kafka. In this way, they were able to balance the amount of data to be processed by the stream and batch application. In the last step, Apache Chukwa was completely replaced by Apache Kafka.

[47]`http://chukwa.apache.org`
[48]`https://hive.apache.org`
[49]`https://pig.apache.org`
[50]`https://cassandra.apache.org`
[51]`https://www.oracle.com/de/mysql`
[52]`https://github.com/Netflix/EVCache`
[53]`https://kafka.apache.org`

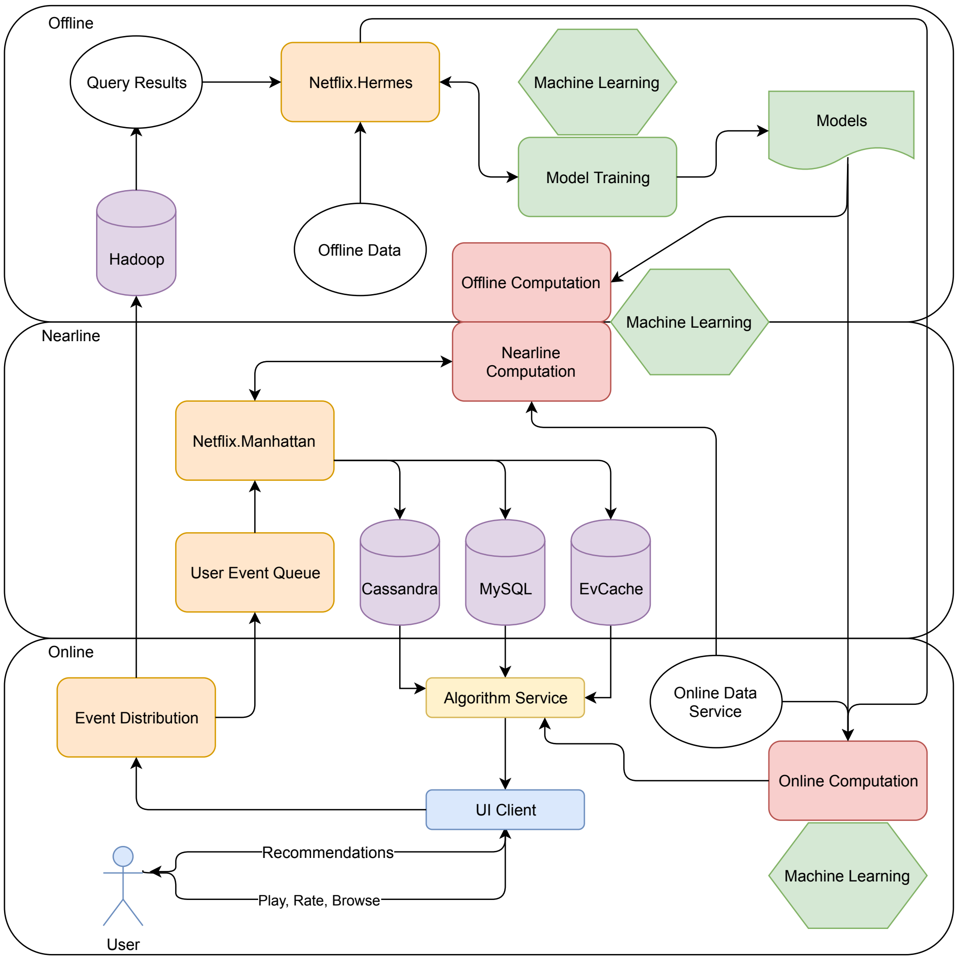


Figure 2.4: The Netflix architecture, based on [53].

Afterward, Apache Kafka provides the data to a router, distributing the data to the appropriate application.

In the last years, the demand to deliver recommendations in real-time has grown. For instance, Netflix aims to generate recommendations as soon as possible to its customers. In this regard, a fast feedback loop has to be implemented [50]. Therefore, Netflix uses an architecture combining Apache Kafka, Apache Spark, and Apache Cassandra to generate personalized recommendations for the “Trending Now” row, as explained in [50]. To achieve this, Netflix first collects data generated in its Impression and Viewing History Service. In the case of “Trending Now”, Apache Kafka distributes the data to Apache Spark to compute currently trending movies and series. The collected trends are stored in the wide column database system Apache Cassandra in half-hour intervals. Additionally, the underlying data model represents metadata information, such as the location, language, and so on. For the final model training by Apache Spark, the trending information is enriched by data of the Viewing History Service and Ratings.

2.8.2 Mendeley Suggest Architecture

Mendeley[54] is a service for researchers to manage their digital library of scientific papers in a convenient manner. In doing so, Mendeley gains knowledge about the papers a researcher is interested in or focusing on to generate personalized recommendations. This alleviates the burden for researchers to find relevant publications in their research field. According to [35], the number of researchers using the service was over 8 million in 2018. In addition, over one hundred million articles are available [BCT18, p. 612].

In [44], a recommender system implementation is divided into five core components. As explained in the article, a recommender system comprises data collection and processing, recommender models, post-processing, online modules, and a user interface. The data collection and processing component transfers the raw data into a valuable representation. Based on this data, recommender models are build, which enable to generate recommendations to the users. Post-processing checks the recommendations for their meaning and plausibility. The online modules are responsible for providing recommendations to the users and keep track of their usage. Finally, the recommendations are presented to the user via a graphical user interface.

The implementation of the recommender system based on these components is elaborated in [45] and illustrated in Figure 2.5. The user interfaces, Mendeley suggest, email, and the newsfeed, serve as the primary sources to collect user events such as clicks or scrolls. All generated events are stored in an event service and filtered by the ones relevant for recommendations by an Apache Spark cluster. Besides, an article service is available, which takes care of processing the metadata of articles added manually by users. Here, the data is stored in an HBase database and processed by Map Reduce. The last service stores user profiles. All services use an Amazon S3 database to store the processed data in the Avro[55] format.

The recommender models are CF-based, popularity-based, trending-based, and CB-based [45]. Due to millions of available publications, a user-based approach was preferred to an item-based one [BCT18, p. 605]. Here, the user-based CF approach utilizes Apache Mahout and Apache Spark [45]. The popularity and trending-based approaches generate recommendations by considering the current activities of the users. At this point, Apache Spark is used. The CB approach is based on the data extracted of the article content and stored in Elasticsearch for retrieval [45]. In all cases, the generated recommendations are forwarded as Spark RDD files to the post-processing module.

The post-processing module filters the recommendations, for instance, by re-ranking them to avoid showing the results over again. Furthermore, some automated processes are part of the post-processing to assure that users receive a sufficient number of recommendations.

In the online module, the final recommendations are stored as JSON files in HBase and Elasticsearch [BCT18, p. 613]. Moreover, Elasticsearch stores the metadata of the publications. The online filter in the online module checks the recommendations based on the most recent information and provides them to the recommendation service.

The user interface is the last module of the architecture. It fulfills two tasks. On the one hand, it collects user events and sends them back to the recommender system such that it constantly

[54]https://www.mendeley.com
[55]https://avro.apache.org

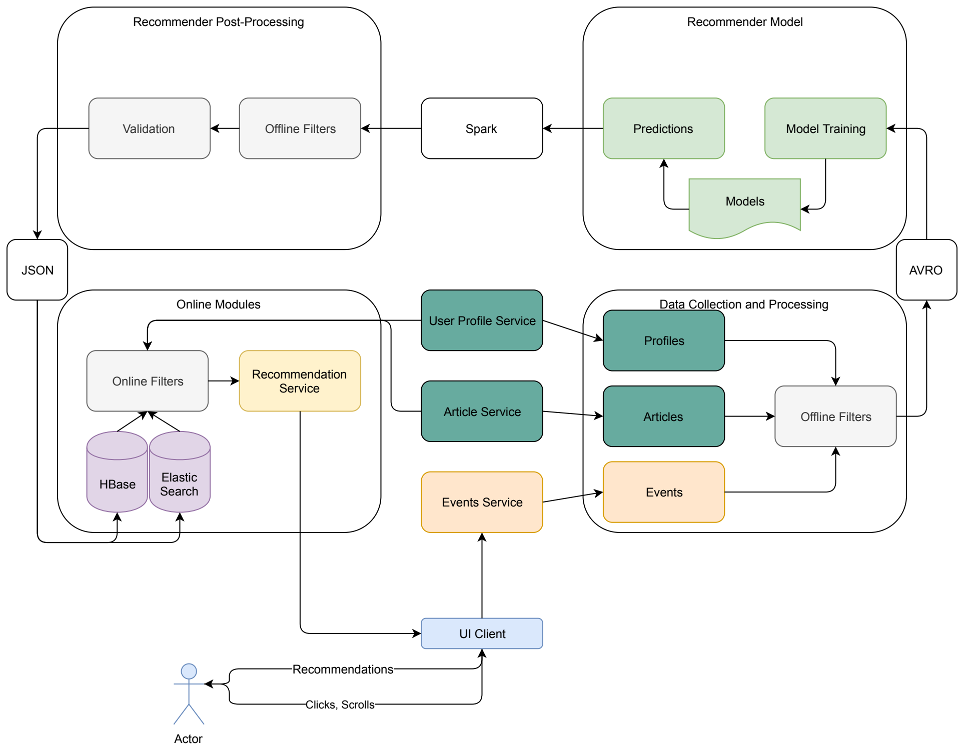


Figure 2.5: The Mendeley architecture, based on [45].

improves the generated recommendations. On the other hand, it provides recommendations to the users.

In order to improve the recommendation results, the CF approach was adapted by significance weighting [BCT18, p. 606-607], time decay [BCT18, p. 607], impression discounting [BCT18, p. 607-608] and dithering [BCT18, p. 608]. Significance weighting aims to scale the impact of users considered as similar but only have a small number of publications in their library. Time decay emphasizes the importance of recently added articles to the library. Impression discount considers presented recommendations that were ignored by a user over a certain amount of time as an indication of having no interest. On the other hand, dithering randomizes the order of recommendations to convey the impression of freshness.

The architecture was evaluated online as well as offline. In the offline evaluation, accuracy metrics such as precision, recall, F1 score, and Mean Average Precision (MAP) were applied [BCT18, p. 617]. For online evaluation, Mendeley used A/B testing, considering the CTR as a metric. Moreover, the CTR was evaluated by considering different aspects, for instance, user interface adaptions and algorithmic changes.

2.8.3 Zalando Recommender System

Zalando[56] belongs to one of the most famous online retailers for fashion in Europe [Fre17, p. 1], [BCT18, p. 687]. According to statistics provided by Statista [9], Zalando had over 31 million

[56]`http://www.zalando.de`

active users in the fourth quarter of 2019. In this sense, a well-implemented recommender system plays an important part role for the company to retain customer satisfaction and engagement [Fre17]. To achieve this goal, the recommender system of Zalando addresses to support the following three technical dimensions: adaptation to new use cases, maintenance costs and complexity, leveraging existing and incorporating new user signals and sources [Fre17].

Depending on the current user context, Zalando combines different recommendation approaches with different levels of personalization. For instance, on a product details page, items are shown similar to the currently visible one, without considering any personalization. In this case, the similarity of two products is determined by their feature vectors based on a scoring function [BCT18, p. 689]. Parts of the feature vector might be attributes such as the color or price of the products. On the Zalando home page, generic personalized recommendations are generated based on similarities between the user and item feature vectors [BCT18, p. 691]. In this case, a user vector might contain attributes that show a preference of the user for a specific brand. The last approach combines both approaches in the sense that the current user context is considered to find similar items, which also match the user preferences described by its feature vector [BCT18, p. 692].

The architecture of Zalando's recommender system is deployed on the Amazon Web Services (AWS) platform and basically comprises offline jobs as well as web services [BCT18, p. 696]. Here, offline jobs are implemented by Apache Spark, whereas the data storage utilizes S3 databases, as illustrated in Figure 2.6. The user logs are the most important data source for the recommender system. Therefore, the user interactions are processed to generate event histories, which are used to extract item and user features such as the number of product clicks and purchases or a user's activity, brand, and price preferences, respectively [BCT18, p. 697]. In the following, a Learning to Rank approach utilizes the features on a daily basis to generate recommendations depending on the different contexts [BCT18, p. 697].

The web services provide top-N recommendations based on an item, a user or their combination via a REST API [BCT18, p. 698]. In order to retrieve user and item features in real-time, the data is indexed in an Apache Solr cluster, which comprises a master and slave instance. To separate recommendation generation and post-processing, requests are processed by a so-called Reco-Servlet, which takes care of filtering and rendering the recommendations, and a backdoor engine, which takes care of their retrieval. The backdoor engine relies on the indexed data on the Apache Solr database and the Learning to Rank model. Additionally, the backdoor engine has direct access to the article features database since they are more stable than user features.

The applied overall evaluation process of the recommender system is based on the offline and online parts. The offline evaluation aims to reduce the number of possible approaches for the following online evaluation performed by A/B testing considering measures such as CTR and CR [BCT18, p. 707]. The offline evaluation was applied to a time-based approach, which utilizes data from the last seven days as training data to predict the purchases of the following day. The used metric was nDCG [BCT18, p. 704].

The authors conclude their architectural description by emphasizing their importance in the following statement: "Focus on operational excellence should never be missing from a scientific investigation of recommender systems" [BCT18, p. 708].

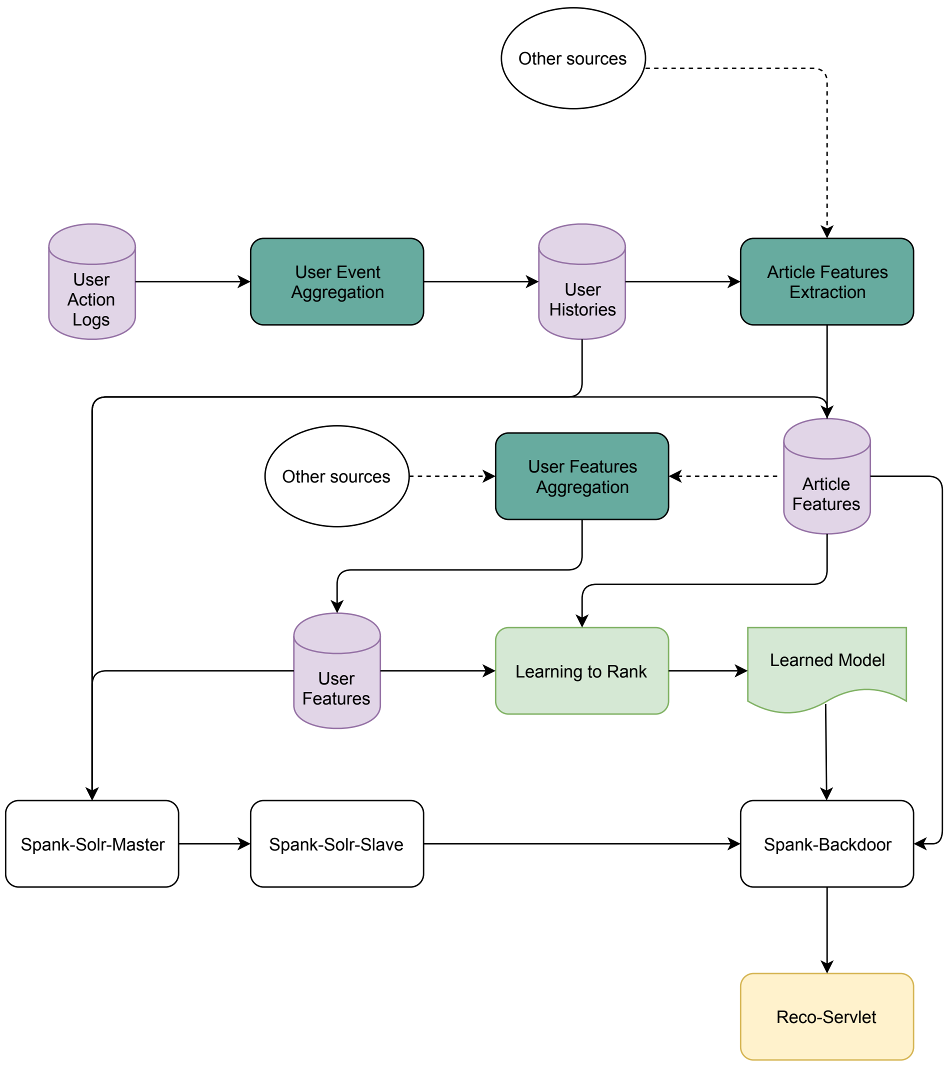


Figure 2.6: The Zalando architecture, based on [Fre17].

2.8.4 Summary

The industrial recommender system implementations of Netflix, Mendeley, and Zalando clarify that a deployed recommender system comprises more than algorithms. Although the application domains of the three implementations differ, the implementations share similarities, which are summarized next. An overview is given in Table 2.2.

Signals

All implementations are based on signals generated by a services' users. For instance, Netflix collects *play*, *rate* and *browse* signals. In case of Mendeley the signals are *clicks* and *scrolls*. In [Fre17, p. 257], Zalando mentions *purchases* and *clicks*. User applications primarily collect these signals.

Recommendation Service

The recommendations have to be provided to the users when requested. For this purpose, these implementations include a recommendation service, which they denote differently. Netflix uses the term *Algorithm Service*, whereas Mendeley referees to it as *Recommendation Service* and Zalando calls it *Reco-Servlet*. However, neglecting the different namings, the purpose of the service is the same.

Post and Preprocessing

Post and preprocessing are explicitly mentioned in the Mendeley architecture but are certainly also parts of Netflix's and Zalando's implementation. For Mendeley, preprocessing aims to filter the collected signals, for instance, by their type and importance. Complementary postprocessing considers the generated recommendations before they are provided to the users to check their plausibility.

Model Training and Models

To provide recommendations to their users, the implementations comprise model training processes based on the application of ML approaches. The models exploit the collected signals and additional information, such as user profiles and item information. For instance, Mendeley incorporates *profiles*, *articles*, and *events*. Zalando integrates *user features* and *article features*.

Data Storage

Data storage systems play an integral part in the implementations and serve two purposes. The first purpose is to store the collected user signals and the second to store precomputed recommendations for the users. In this regard, various types of data storage systems are part of the implementation to balance their strengths and weaknesses, such as NoSQL databases and relational databases.

Data Collection

Data collection aims to provide the collected user signals to the system. Therefore, the implementations contain processing systems to distribute the user signals to the responsible components. In the Mendeley architecture, this is part of the *Data Collection and Processing* component, whereas in the Netflix case, the *Event Distribution* and *User Event Queue* components take care of this task.

Online and Offline Layer

Considering the architecture of Netflix, it relies on computational expensive models build in the *Offline* layer but also on the most recent information build in the *Nearline* and *Online* layers. The overall architecture comprises a batch and streaming pipeline. In this sense, it can be considered as a Lambda architecture. It aims to balance historical and recent information to generate user recommendations.

Table 2.2: Comparison of the industrial recommender systems regarding their application domains, storage, and processing systems as well as their applied recommender system approaches and layers.

	Netflix	Mendeley	Zalando
Application Domain	E-resource service	E-library	E-commerce
Storage	Cassandra MySQL EvCache Hadoop	HBase Elasticsearch	AWS S3
Processing	Netflix.Hermes Netflix.Manhattan Pig Hive Kafka Spark	Spark Hadoop	Spark
File formats		JSON AVRO RDD files	
Approaches	MF Restricted Boltzmann Machines	Learning to rank	CF CB Popularity Trending
Layers & Components	Offline Nearline Online	Data Collection and Processing Recommender Model Post Processing Online Modules User Interface	Data and Jobs

2.9 General Trends and Future Developments

Considering the accepted papers on the recommender system conference RecSys in 2020, one exciting research trend is bias. Furthermore, a keynote given by Ricardo Baeza-Yates was dedicated exclusively to this topic [BY20]. In [STSO20, p. 378], the authors investigate the causal effect of purchases which are based on recommendations and products a customer would have purchased anyway. The authors of [ZHZC20, p. 551] aim to generate unbiased recommendations based on implicit user feedback since such feedback might not reflect a user's interest in all cases. The bias on so-called satisfaction surveys is investigated in [CTP$^+$20, p. 450]. In such surveys, a user's explicit opinion about a certain item is asked but might already lead to a bias by participation in the survey [CTP^{+}20, p. 451]. The bias in synthetically generated data is investigated in [HOdRvH20, p. 190]. For this, the authors also developed the Simulator for OFfline leArning and evaluation (SOFA). The bias based on a user's personality and the influence of the personality on the recommendations is considered in [MZS20]. Here, the focus is, particularly on music recommender systems. Another exciting aspect, examined by the authors of [YLH^{+}20], is the bias generated by the position of recommendations.

Besides bias, fairness and explainability are researched aspects of recommender systems. In this context, the fairness of a recommender system is given when it provides consistent results among various groups, for instance, male and female [AMBM20, p. 726]. In the paper, the authors examine fairness by a recommender system's capability to generate recommendations based on the proportion of a user's preferences [AMBM20, p. 727]. Explainability aims to provide users with information about why certain items are recommended to them. For instance, in [TG20, p. 462], the authors investigate explanations based on repeatedly consumed items. On the other hand, in [BFCK20] explanations of novel items are investigated by identifying different so-called personas in a user's profile. For instance, one persona covers a user's interest in horror and the other in comedy.

Furthermore, deep learning approaches aim to increase the accuracy of recommender systems. A comprehensive overview of the topic is given in [ZYST19]. The authors consider deep learning as a possibility to learn non-trivial user-item relationships. On the other hand, in [FMY^{+}19], a deep learning approach utilizing social network information is introduced. For instance, in [KH17] the importance of deep learning for recommender systems is explained by its capability to reuse established MF approaches. Nevertheless, in [DCJ19] several deep learning approaches have been analyzed and discussed. One result of the work is that less complex algorithms can partially outperform these approaches.

From an industrial perspective, companies still have to tackle the challenge of providing recommendations on historical and most recent data, as discussed in Section 2.8. In [RSP20], a Japanese dating company, explains its hybrid approach based on batch jobs and streaming endpoints to generate recommendations for warm-up and cold-start users. Besides, this shows the architectural need to investigate incremental recommender systems approaches further.

2.10 Discussion

This chapter has provided an overview of common approaches in the research field of recommender systems. It further has discussed the evaluation of recommender systems considering what to measure and how to measure. In this regard, libraries and frameworks have been presented which support the evaluation process by providing implementations of popular and established approaches. They enable a systematic evaluation of these algorithms on given data sets. In particular, they are useful to run hyperparameter optimization, for instance, a grid search, to find optimal parameters for a specific algorithm and data set. However, their main focus is based on implicit or explicit user feedback, like user ratings or customer purchases. In this sense, they are well suited to analyze the data for one specific user signal type. Nevertheless, in reality, users generate signals of various types. These libraries and frameworks do not directly support the aspect of multiple signal types during the evaluation of data sets and algorithms.

In [JMO19], the authors consider this aspect and investigate how to combine implicit and explicit user signals in a unified approach. Their approach reflects the thesis' idea to incorporate various channels and user signals in the recommendation process. However, this thesis especially investigates the combination of multiple sources of implicit and explicit user signals. In this regard, it does not discriminate whether an algorithm is targeting rather implicit than explicit user signals. In [WM18], the authors consider user signals as a monotonic sequence. This consideration means they assume a specific order in the occurrence of user signals. For instance, a user first clicks, then purchases, and finally reviews a product. Such a sequence might be a limited view of possible sequences since it assumes a specific user behavior. This thesis does not make such assumptions since it is also reasonable for a customer to review a product without having bought it. At this point, the monotonic chain represents one data aggregation type. In contrast, the thesis' idea is to propose a generalized definition of data aggregation types adapted to specific application domains and shared to get further insights.

The FAiR framework also includes this thesis' idea to evaluate a recommender system with different metrics and support the results' comparison. However, a user of FAiR has to run the algorithms by himself or herself. Besides FAiR, RiVal also focuses on the evaluation process and not the algorithms. Idomaar reflects the idea of providing an infrastructure for a recommender system and can be considered as a complement to the proposed concept. However, its implementation is technology agnostic and not easy to set up. Although these frameworks are beneficial for evaluating recommender systems, they do not focus on multiple user signal types.

Considering the frameworks sRec, StreamingRec, StreamRec, and AlpenGlow, they focus on evaluating streaming recommender systems. For instance, StreamingRec considers the application domain of news articles. StreamRec aims to generate fast real-time recommendations by optimizing item-based CF. AlpenGlow also evaluates online recommender systems. SRec targets to provide real-time recommendations but focus on explicit user signals. However, in real-world implementations, a recommender system has to handle user signals of various types.

3 Fundamentals of Benchmarking

The objective of this chapter is to provide an overview of the topic of benchmarking and Big Data benchmarking in particular. Initially, a motivation to investigate benchmarking in the context of recommender systems is provided. Then, the history of benchmarking in general and in the specific area of Information Technology (IT) is given. This leads to a consideration of the different benchmark types, such as micro, component, system, application, and end-to-end. In the following, the concepts and ideas of consortia such as the Transaction Processing Performance Council (TPC), Standard Performance Evaluation Corporation (SPEC), Linked Data Benchmark Council (LDBC), and BenchCouncil to standardize the benchmarking process for transparency are introduced. After that, features and requirements which characterize a well-defined benchmark, e.g., relevance/acceptance, portability, scalability/extensibility/adaptability, simplicity/usability, repeatability/verifiability/reproducibility, and fairness, are defined. Accordingly, standard terms and definitions, such as System under Test (SUT), data models, workloads, queries, and metrics, are explained to assure a consistent understanding of the topic. Finally, current state-of-the-art Big Data benchmarks are presented and investigated considering their application domains, use cases, workloads, data, and metrics.

3.1 Motivation

As discussed in Chapter 2, recommender systems play an important role in academia and industry alike. In the context of academic research, the focus is mostly on evaluating and comparing different algorithmic approaches in view of their accuracy, whereas in industry, the integration of these approaches into a system environment is another important aspect to consider. This difference occurs because, from an industrial perspective, a recommender system is characterized by more than just plain algorithms. Instead, it is a composition of components and modern technologies, as shown in Section 2.8. This composition of components requires a complex technical selection process, which can be supported by a benchmark. For instance, the recommender systems of Netflix, Mendeley, or Zalando have to collect, store, and process many user interactions in a fast manner. In this regard, the involved technologies have to deal with large data volumes and a still increasing velocity of data generated by users. Furthermore, with the integration and incorporation of various data sources, technologies have to be able to exploit as much information as possible to increase, for instance, recommender system results. Consequently, the applied technologies have to meet high requirements for data storage and data processing.

To support and facilitate the technology selection process, the use of benchmarks represents an effective tool to verify whether the considered technologies meet the defined requirements. Besides accuracy, this includes non-functional requirements such as scalability or availability, measured by throughput or response time, which are also crucial for the overall performance of a recommender system. Moreover, these requirements highly influence the acceptance of a recommender system by the end users since they assure a fast and smooth user-system interaction. Consequently, this motivates to introduce and explain benchmarks developed with the purpose of evaluating these requirements by low-level metrics in the following.

3.2 History of Benchmarking

3.2.1 Benchmarking Origin

The term benchmark consists of the nouns “bench” and “mark”. According to [Tuc00, p. 71], a mark on a bench served craftsmen during the process of measuring. Benchmarking has its origin in the topography domain to compare heights and directions based on reference points, the benchmarks. In this sense, [Tuc00, p. 71] considers benchmarking as the process of measuring, defining criteria, determining reference values, and comparing the results.

According to [Tuc00, p. 71], [Sta09, p. 8] a starting point of modern benchmarking were the visits of Japanese managers to get insights about the production processes of American companies after World War II.

In the seventies, the American company Xerox was faced with strong competition. In order to stay competitive, Xerox decided to visit companies superior in their field to improve their processes, for instance, the retail company L.L. Bean to get insights about its distribution process [Sta09, p. 9]. Additionally, Xerox analyzed their competitors’ products by reverse engineering to understand how they were constructed [Tuc00, p. 72] and the corresponding production process.

Therefore, in the organizational context, benchmarking is considered as part of an improvement culture as well as a driver and a short-cut for improvement. Furthermore, to solve problems, to build up networks, to justify proposals, and to identify weak points of competitors [Sta09, p. 12-14]. To spot out weak points, to scout out opportunities for improvement, and to plan actions to achieve them are mentioned in [Rol95, p. 211]. In [Sta09, p. 19-20], seven methods are introduced, namely, public domain, one-to-one, review, database, trial, survey, and business excellence models benchmarking.

General Definition

Therefore, in general, benchmarking defines a continuous and systematic process of comparing, for instance, products and services as well as processes and methods to identify weaknesses and initiate actions to eliminate them [52]. From a business and process perspective, Camp et al. ’s definition is widely accepted and used. It says: “Benchmarking is the continuous process of measuring products, services, and practices against the toughest competitors or those companies recognized as industry leaders” [Cam89, p. 89]. According to Rolstadås [Rol95, p. 5], benchmarking can be classified into three main types: competitive, internal, and functional. As the name implies, competitive aims to compare to direct competitors, whereas internal focuses on internal aspects and functional considers a generic comparison across different industries. In this sense, benchmarking in Information Technology can be classified as competitive and internal.

3.2.2 Benchmarking Information Technologies

Nowadays, benchmarking is applied in a wide range of application domains. Besides management, benchmarking is widely used in the computer industry and the information technology domain. The reason for that is based on the fact that computers are basically invented to accelerate and improve task execution by making them easier and faster [Wei90, p. 65]. According to [LC85], the first necessity of benchmarking arose in the sixties due to a diversity of systems and vendors. In this context, the benchmarks serve to compare the capabilities of computer systems primarily considered as "a routine used to determine the speed performance of a computer system" [HJ65, p. 27]. A first approach were the Auerbach Corporation's Standard EDP Reports, which considered, for instance, the data structure, the CPU speed, the storage size as well as the price of the systems [LC85, p. 8]. EDP stands for Electronic Data Processing. In the following, benchmarks evolved to consider an applied point of view, in a sense that the routines target particular applications [LC85, p. 8-9].

Besides, the first idea of synthetic programs to mimic real application was introduced by Buchholz [Buc69]. Such a program aims to emulate common data processing tasks, like the file maintenance program implemented by Buchholz [LC85, p. 10]. As stated by [Wei90, p. 66]: "The Wheatstone benchmark was the first program in the literature explicitly designed for benchmarking" and published in 1976. Nevertheless, it must be noted that the benchmark was not intended to execute useful operations [WCZ$^+$16, p. 66]. Another important program in this regard is Linpack, which was initially rather published to perform linear algebra subroutines than to serve as a benchmark [Wei90, p. 67]. Other notable benchmarks published in the eighties are the EDN benchmarks (1981), Dhrystone (1984), Livermore Fortran Kernels (1986), Rhealstone (1989) and the ones developed by the Standard Performance Evaluation Corporation (SPEC) in 1989 [Wei90, p. 71-72]. Overall, these benchmarks were basically targeting computer system performance.

3.2.3 Benchmarking Database and Big Data Systems

In the domain of DBMSs, the necessity for benchmarks was driven by the demand for high-performance transaction systems, for instance, Automated Teller Machine (ATM) and automatic operations in gasoline stations as stated by Omir Serlin in [Ser91, p. 1]. In 1983, the Wisconsin Benchmark [BDT83] was published to evaluate the performance of databases specifically. Later in 1985, the TP1 was developed by IBM [NLW$^+$09, p. 2] and DebitCredit by Jim Gray [ABB$^+$85]. However, these benchmarks did not provide any standards for applying them, which led to arguable results. For instance, vendors tried to optimize the results to promote their products better. This procedure was denoted as "benchmarketing" [NLW$^+$09, p. 2]. Nevertheless, according to [VM09], these benchmarks represent starting points to found non-profit consortia to specify standardized benchmarks such as the Transaction Processing Performance Council (TPC). Based on the TP1 and DebitCredit benchmarks, the first benchmark published by the TPC was the TPC-A in 1989. Consequently, more benchmarks have been developed and standardized by the TPC.

With the advent of the Big Data era around 2005, new challenges arose from the possibility to collect and analyze data for further decision making. The availability of the data is based on the spread of the Web 2.0, especially social media, which enables users to participate actively on the Internet instead of only consuming information. This is further increased by the popularity and usage of mobile devices, which enable to access content from anywhere at any point in time. Hence, Big Data is basically characterized by the 3Vs: volume, velocity, and variety [Lan01]. Volume defines a large amount of data, velocity defines the speed at which the data appears and gets processed, and variety considers the different data representations. To tackle the characteristics of Big Data, new technologies have been developed. For instance, databases evolve from monolithic to distributed systems to handle large amounts of data as well as to increase availability. An example of this are NoSQL databases. They relax classical requirements on databases, which guarantee Atomicity, Consistency, Isolation, and Durability (ACID) of transactions by the concept of BASE. BASE stands for Basically Available, Soft state, and Eventual consistency and enables databases to distribute their data among different servers [Cat10, p. 1]. Besides, they further enable to manage various kinds of data. To deal with the high velocity of data in real-time, sophisticated processing technologies emerged. Considering this, benchmarking Big Data systems rather means to evaluate a pipeline of connected components than a single system.

3.3 Benchmarking Types and Consortia

3.3.1 Types

In Big Data benchmarking literature, various types of benchmarks are mentioned, applied, and developed. Therefore, this section introduces and explains the different benchmarking types in this context. In the review of Han et al. [HJZ18], benchmarks are classified into micro benchmarks, end-to-end benchmarks, and benchmark suites. The benchmark compendium of Ivanov et al. mentions a similar classification [IRP^{+}15]. In [Bog14, p. 49-50], benchmarks are additionally classified into hardware and software benchmarks. Hardware benchmarks are further divided into component, system, and micro benchmarks, whereas software benchmarks are divided into application and system software as well as micro benchmarks. In [BWT17, p. 63-64] benchmarks are considered as application-driven or micro. In the following, the different benchmark types are listed and explained.

Micro Benchmarking

A micro benchmark aims to evaluate a small and specific functionality of hardware or software. In the case of software, this functionality might be a specific method or the implementation of an algorithm [Bog14, p. 49-50]. In [IBGZ18, p. 2], for instance, the authors mention different streaming functionalities as part of a micro benchmark. In the case of hardware, the performance of a particular CPU unit could be part of the benchmark.

Component Benchmarking

A component benchmark mostly serves to assess the performance of hardware components such as CPU, storage, or network devices [Bog14, p. 49-50]. Nevertheless, software could also be considered as a component such that the differentiation into component benchmark and application software benchmark made in [Bog14] is neglected here. Examples of software components are business software or mail clients [Bog14].

System Benchmarking
Complementary to a component benchmark, a system benchmark evaluates a system as a whole, considering it as a black box [Bog14, p. 49]. In this sense, such a benchmark evaluates a specific and dedicated functionally of a system, for instance, providing a database or a DBMS.

Application Benchmarking
The goal of an application benchmark is to evaluate a system considering real-world scenarios or realistic use cases [HJZ18, p. 582], [IS18, p. 1]. For Poess et al., an application benchmark also has to simulate a real use case [PRJ17, p. 574].

End-to-end Benchmarking
End-to-end benchmarks aim to evaluate the performance among various components build on each other, for instance, considering hardware components as well as operating systems an application is running on. In this case, the performance from one end, the application, to the other end, the hardware, is evaluated. Another example represents the evaluation of a specific software technology stack.

Benchmarking Suite
In contrast to end-to-end benchmarks, a benchmark suite is a composition of different benchmarks [HJZ18, p. 582]. More precisely, a benchmark suite covers various application domains or scenarios a benchmark user can choose from.

3.3.2 Consortia

Regarding benchmarking of information systems and technologies, four established and widely accepted consortia exist that pursue the goal of providing standardized guidelines and specifications for an objective comparison among vendors. Therefore, in the following, the four consortia, namely, the Standard Performance Evaluation Corporation (SPEC), the Transaction Processing Performance Council (TPC), the Linked Data Benchmark Council (LDBC), and the International Open Benchmark Council (BenchCouncil) are introduced.

Standard Performance Evaluation Corporation (SPEC)
The SPEC was founded in 1988 by vendors to provide a trustful platform for realistic benchmarks and prevent benchmarketing, which was very common at that time. According to [17], the SPEC is "a non-profit corporation formed to establish, maintain and endorse standardized benchmarks and tools to evaluate performance and energy efficiency for the newest generation of computing systems". The SPEC provides a wide range of benchmark specifications covering cloud platforms, CPUs, graphics and workstation performance, handhelds, high-performance computing, java client/server applications, mail servers, storage, power, and virtualization [17]. In order to guarantee high-quality benchmarks, the SPEC is organized into subcommittees focusing on the different benchmark specifications. To facilitate the configuration and setup process of their benchmarks, the SPEC provides platform-independent tools.

Transaction Processing Performance Council (TPC)
Founded in 1988, the TPC is "a non-profit corporation focused on developing data-centric benchmark standards and disseminating objective, verifiable performance data to industry" [21]. In contrast to the SPEC, the TPC has a clear focus on benchmarking transaction processing and DBMSs. As a consequence, the benchmarks provided by the TPC are covering data integration, decision support solutions, online transaction processing, and virtualization. In this regard, the benchmarks represent real-world applications by simulating a wholesale supplier, an online retailer, or a brokerage company. To achieve high credibility, all provided test sponsor results are reviewed by an independent auditor before publication.

Linked Data Benchmark Council (LDBC)
The LDBC aims to specify "benchmarks, benchmarking procedures and verifying/publishing results for software systems designed to manage graph and RDF data" [22]. In contrast to the TPC and the SPEC, the LDBC targets to benchmark RDF and graph database systems, which get active attention in recent years. In particular, the benchmarks support transaction, analytic, and graph analytical queries. Similar to the TPC, the LDBC results are audited before they are published publicly.

International Open Benchmark Council - BenchCouncil
BenchCouncil "is a non-profit research institute which aims to promote the standardization, benchmarking, evaluation, incubation, and promotion of open-source chip, AI, and Big Data techniques" [1]. BenchCouncil provides benchmarks covering Big Data and Artificial Intelligence (AI) as well as a Big Data generator. The development of a benchmark comprises a six-step process: (1) propose a benchmark, (2) found a working group, (3) publish benchmark specifications, (4) publish implementation specifications, (5) organize challenges and (6) present results [1]. Moreover, the council organizes workshops and conferences to further support development in the topic of benchmarking.

3.4 Requirements on Benchmarks

To develop a well-defined and accepted benchmark, some requirements have to be met. According to Jim Gray [Gra93], a benchmark has to be relevant, portable, scalable, and simple. Furthermore, Huppler [Hup09] characterizes a good benchmark by its relevance, repeatability, fairness, verifiability, and economy. For the TPC a benchmark needs to be relevant, understandable, portable, and scalable. Besides, good metrics, coverage, and acceptance are preferable [46]. The SPEC specifies four design objectives: portability, repeatability and reliability, consistency and fairness, and application-oriented [34, p. 15]. In [HLX14] additional requirements considering Big Data such as adapting to different data formats, portability to representative software stacks, fair measurement, extensibility, and usability are mentioned. In the following, the collected requirements are listed and described.

Relevance and Acceptance
A benchmark has to be relevant in the sense that its results lead to new insights for academia or industry [Hup09]. For Huppler, relevance is given when the benchmark targets software

features and hardware systems used in the real world [Hup09]. In this sense, relevance covers one design goal of SPEC, that a benchmark has to be application-oriented. Furthermore, relevance includes broad adaptability and longevity [Hup09]. Jim Gray also considers as relevant to measure the peak performance of a system [Gra93]. In this regard, meaningful and understandable metrics are important [Hup09]. Since only a relevant benchmark will gain acceptance by vendors and researchers, both requirements are highly interconnected.

Portability
Gray considers portability to implement and deploy the benchmark for and on various systems [Gra93]. According to [Hup09, p. 26], due to current standards in programming and query languages, portability is considered non-critical. However, for benchmarking Big Data systems, which might be built on multiple technologies, portability has to be considered [HLX14, p. 7]. In this sense, it should be preferred to implement a benchmark in a high-level language such that a wide audience is able to apply it easily.

Scalability, Extensibility and Adaptability
In [Gra93], scalability is defined as the ability of a benchmark to be applied to small and large systems as well as parallel and distributed ones. For instance, in the context of Big Data, scalability refers to the data generation process of a benchmark to increase the volume or velocity of the data considering the tested software or system [HLX14, p. 11]. In this sense, scalability additionally enables a benchmark to support future developments. Furthermore, modern Big Data benchmarks should be extensible and adaptable to support, for instance, new workloads or data sets [HLX14, p. 8].

Simplicity and Usability
Simplicity involves the deployment, configuration, and execution of a benchmark as well the representation and interpretation of its results [HLX14, p. 8]. In this sense, the benchmark must be simple in its usage and providing understandable results [Gra93]. At this point, the usability and user experience could be improved by a convenient user interface [HLX14, p. 8]. In addition, since simplicity reduces the potential effort to apply and run the benchmark, it is more likely to be economical in its application regarding, for instance, cost-efficiency and execution time [Hup09]. However, a benchmark should not oversimplify the tested scenario, referred to as coverage by the TPC. In doing so, a benchmark gains more acceptance in the industry and academia alike.

Repeatability, Verifiability and Reproducibility
According to [Hup09], repeatability ensures that the benchmark provides the same results when running multiple times under the same conditions. In this sense, repeatability increases the reliability and confidence in the results of the benchmark [Hup09]. Furthermore, the repeatability of a benchmark is preferable since it additionally enables audits or third parties to reproduce and verify the provided results. Accordingly, the benchmark configuration has to be documented in a traceable manner to rerun the benchmark [HLX14, p. 7]. This involves a precise specification of the involved components.

Fairness
Huppler considers fairness as the possibility for all compared systems to participate equally in the benchmark [Hup09]. Consequently, fairness assures that no participant of the benchmark benefits from specific workloads. Han et al. [HLX14] mention configuring the competing systems, especially Big Data systems, in a fair and appropriate way, for instance, by adapting the default parameters such that the systems become comparable.

3.5 State-of-the-Art Benchmarks

This section provides an overview of current state-of-the-art benchmarks and Big Data benchmarks in particular selected based on the following considerations. First, famous and widely used benchmarks published by the four benchmarking consortia (see Section 3.3.2) are explained. Second, micro and application benchmarks covering storage and processing are analyzed. Each benchmark is presented with a focus on the following aspects: data model, data, workloads, metrics, technologies, design, and code. Data models are considered when they are explicitly described. Depending on the benchmark, they are based on specific data sets, generate synthetic data on their own, or use a combination of both. Therefore, the aspect data comprises these three possibilities.

3.5.1 TPC-DS

The TPC-DS benchmark aims to evaluate the performance of decision support systems [31]. In contrast to its first version, the current one targets to analyze the performance of SQL-based Big Data solutions [PRJ17]. Therefore, some adaptions to the first version were made to cover these new requirements. One of the major differences between the first and second version of the benchmark refers to the new requirements which arose with the advent of Big Data moving from ACID-based systems to more relaxed BASE-based systems [PRJ17, p. 574]. Additionally, Big Data systems are loosely coupled compared to traditional database systems, where the database system takes ownership of the data [PRJ17, p. 574].

Data Model The underlying data model describes a sales organization distributing its goods in stores, by catalogs, and over the Internet [29, p. 22]. Overall, the model comprises 17 dimensions and seven fact tables, such as store sales, catalog sales, and web sales, as well as their corresponding returns and inventory [29, p. 22].

Data The raw data for the benchmark is generated in the form of flat files by the provided dsdgen tool. A scale factor specifies the raw data size in GB, for instance, 100, 300, 1,000, 3,000, 10,000, 30,000 or 100,000 [PRJ17, p. 576]. Afterward, the generated files have to be loaded into the data processing system by using tools or APIs [29, p. 71].

Workloads The benchmark comprises 99 queries classified into reporting, ad hoc, iterative Online Analytical Processing (OLAP), and data mining [29, p. 19]. Typical queries are, for instance, calculate the average sales quantity of an item or select the products generating the highest revenue for a given year or month. To generate the queries, the dsqgen tool is applied. The execution process of the benchmark consists of the following tests: a load test, a single user test, two multi-user tests, and two data integration tests, whereas T_L, T_{SU}, T_{MU} and T_{DI} denote the execution times of the corresponding test. The repetition of the data integration test tends to evaluate the update process of the fact tables, e.g., adding or deleting sales data [PRJ17, p. 576].

Metrics The overall performance metric for the benchmark is determined as the geometric mean of the four executed tests as [PRJ17, p. 576]:

$$QphDS@SF = \left\lfloor \frac{SF \cdot (S_q \cdot 99)}{\sqrt[4]{(T_{SU} \cdot S_q) + (T_{MU_1} + T_{MU_2}) + (T_{DI_1} + T_{DI_2}) + (0.01 \cdot S_q \cdot T_L)}} \right\rfloor$$

where SF specifies the scale factor of the generated data and S_q the number of concurrent users involved in the benchmark process. Additionally, a price-performance metric is defined as the ratio of the system's costs to its performance.

Technologies In [PRJ17, p. 576-584], the authors applied the benchmark on four different systems without revealing the underlying technologies. However, the four tests cover two HDFS-based storage engines as well as one representative of a traditional Relational Database Management System (RDBMS) and one representative of a columnar in-memory storage system. In this regard, results of the execution of the single [PRJ17, p. 578-579] and multi-user tests [PRJ17, p. 580] as well as resource tests [PRJ17, p. 581-584] considering CPU, IO, memory and network utilization are provided.

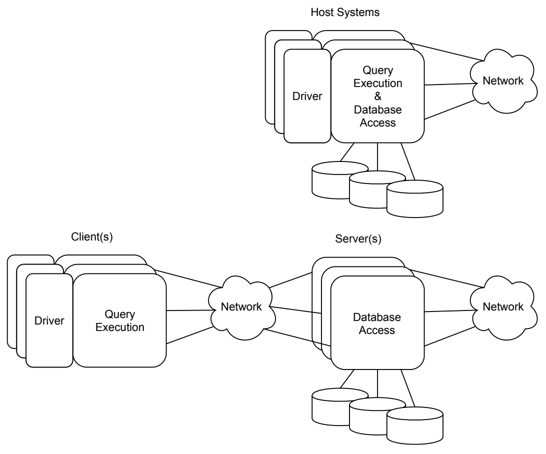
Design As illustrated in Figure 3.1a, the benchmark driver, short driver, takes care of the query execution and database access during the benchmark execution process. Here, two configurations are possible: a host-based and a client-server-based one. In the host-based case, the driver is located on the same machine as the SUT. In the client-server-based case, both run on separate systems. To assure the communication between the driver and the SUT, it might be required to develop an implementation-specific layer as depicted in Figure 3.1b.

Code All files to execute the benchmark are available on the TPC download page [30].

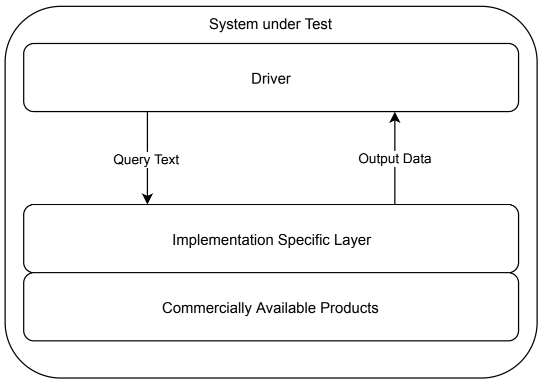
3.5.2 BigBench

BigBench is an end-to-end benchmark that aims to standardize the assessment of Big Data analytic tasks. The first specification of the benchmark was introduced in [RGH$^+$12] and further elaborated in [RFD$^+$14]. In the following, BigBench became a standardized TPC benchmark, namely, TPCx-BB, as discussed in [CGL$^+$16].

Data Model BigBench's underlying data model represents a retail product supplier in the e-commerce domain. The data model consists of entities that are common for the e-commerce



(a) Two different configurations: a "host-based" and a "client/server" configuration, based on [29].


System under Test
Driver
Query Text
Output Data
Implementation Specific Layer
Commercially Available Products

(b) Implementation of a specific driver layer, based on [29].

Figure 3.1: Different benchmark configurations (left) and a driver specific layer implementation (right).

domain, as illustrated in Figure 3.2. This includes customers and items as well as sales and review information of items. Additionally, the online interactions of customers are available as weblogs. Besides, the different market prices of competitors for the provided items are part of the model. Furthermore, the data model entities represent the various data types in the Big Data context, e.g., structured, unstructured, and semi-structured. For instance, reviews are mostly free-text and, therefore, unstructured data. Weblogs, on the other hand, are considered as semi-structured due to their specific format but missing data type information. Customer, item, and sales information are represented in a structured format.

Data In addition, the benchmark includes a scalable data generator known as Parallel Data Generator Framework (PDGF). In its initial version [RGH+12], BigBench used dsgen and PDGF for data generation. Additionally, a review generator was developed to generate unstructured data, e.g., reviews, using the Markov Chain Algorithm [RGH+12, p. 3]. Since the publication of [RFD+14, p. 2], the PDGF can generate the complete data model. Besides, the data generator is able to produce data with different scale factors, e.g., 100, 300, 1,000, 3,000, 10,000, 30,000, 100,000 GB [RFD+14, p. 5].

Workloads The analytical tasks of the benchmark are defined by 30 queries which cover marketing, merchandising and operations aspects as well as considerations of supply chain and new business models [RGH+12, p. 4]. As a result, the queries cover the different data structures [Raa19, p. 4]. Additionally, the queries cover different algorithms such as statistical, path, and text analysis as well as classification, clustering, and reporting [RGH+12, p. 4]. To provide a technology-agnostic benchmark, the queries are described in natural language such as "find the most frequently sold products" or "find the most viewed products before an online

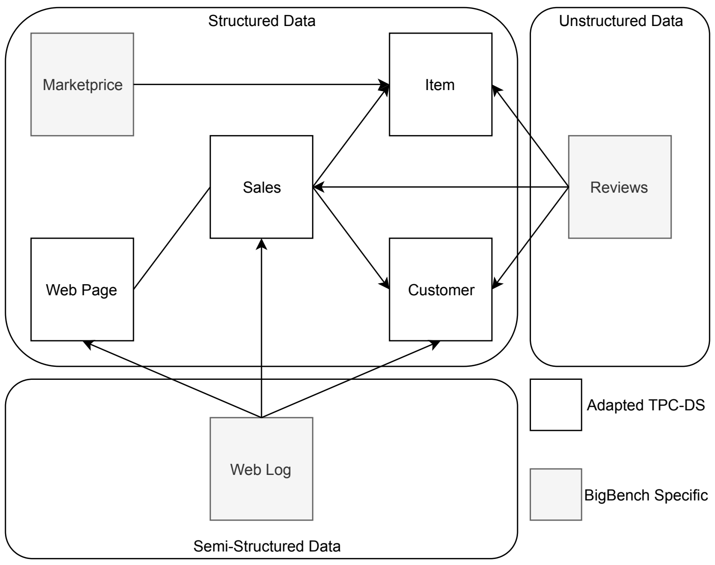


Figure 3.2: TPCx-BB data model, based on [32, p. 19].

purchase" [32, p. 92]. Since BigBench targets batch processing in the first place, in [IBGZ18] the authors propose how to extend it to support stream processing as well.

Metrics The considered metrics are a performance metric and a price-performance metric, defined as queries per minute for a specific scale factor and related to the costs of the SUT, respectively. The overall performance metric for the benchmark is determined as [32, p. 37-38]:

$$BBQpm@SF = \frac{SF \cdot 60 \cdot M}{T_{LD} + \sqrt{T_{PT} \cdot T_{TT}}}$$

where T_{LD} is the load factor, T_{PT} represents the geometric mean weighted by the number of queries and T_{TT} is the average of all throughput test runs considering the number of streams [32, p. 37-38]. Besides, M denotes the number of queries and SF the scale factor. The price-performance metric is defined as the ratio of the system's costs to its performance.

Technologies The first implementation of the benchmark was applied to the Teradata Aster DBMS [RGH$^+$12, p. 2]. In [CRS$^+$13], the authors apply the benchmark on the Hadoop technology stack using Apache Hadoop, Apache Hive, Apache Mahout, and the Natural Language Processing Toolkit (NLTK) for declarative and procedural queries. Furthermore, in [CGL$^+$16, p. 34], the benchmark was applied to Hive on MapReduce, Hive on Spark, Hive on Tez, and SparkSQL. Another implementation, based on Apache Flink, is provided in [BGSZ17].

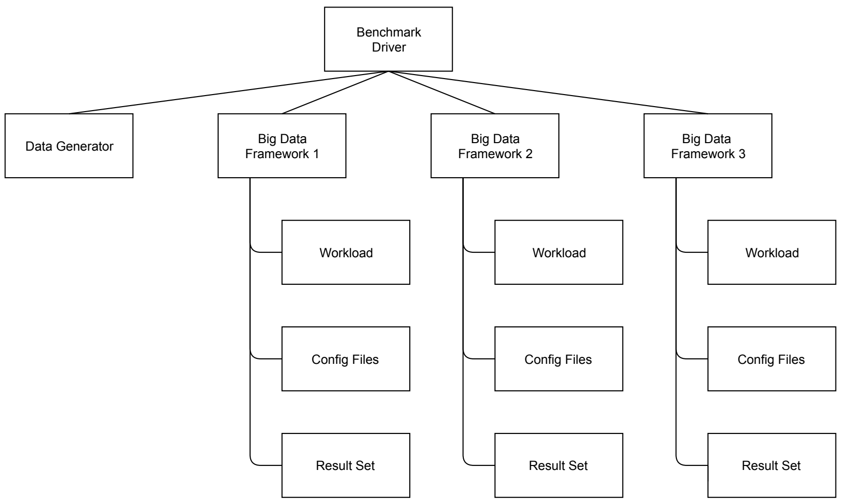


Figure 3.3: An exemplified illustration of the benchmark kit with three Big Data frameworks, based on [CGL+16, p. 27].

Design The overall design of the provided benchmark kit is depicted in Figure 3.3. Here, a framework comprises the Big Data analytics software and its APIs as well as the distributed computing engines and libraries required for the benchmark execution such as MapReduce, Apache Spark, or Apache Flink [32, p. 13], [CGL+16, p. 28]. According to [32, p. 21], the kit contains documentation, configuration files for the SUT, a script for the benchmark execution, the benchmark driver which takes care of the execution, time measurement, and result computation, and scripts to verify and check the results. To reduce the high complexity of running and configuring the benchmark, it is provided as a self-contained kit, which, per default, includes configurations for the following data analytics distributions: Cloudera CDP, Cloudera CDH, and Hortonworks HDP [CGL+16].

Code The benchmark kit containing all relevant components to execute the benchmark requires a registration for its download and is published under the TPC-Tools License Agreement [19].

3.5.3 Graphalytics

Graphalytics is a Big Data benchmark developed targeting the performance of graph-processing platforms [CHI+15a, p. 1]. The first vision of the benchmark and its idea are explained by Capota et al. in [CHI+15a]. Since the publication of [IHN+16], the benchmark is an official benchmark of the LDBC. Besides the LDBC, the SPEC supported its development and implementation [CHI+15a, p. 1].

Data To generate large graphs, the benchmark utilizes the Graph500 data generator [38] and the LDBC Social Network Benchmark (SNB) data generator [27], where the latter uses t-shirt sizes as scale factors to assure an easy understanding [IHN+16, p. 1317].

Workloads Initially, the benchmark prototype consisted of five common scenarios applied to graphs [CHI+15a, p. 4]. In [IHN+16], the authors implemented six scenarios based on a conducted survey. As a result, Breadth-first Search (BFS) and PageRank, as well as identifying weakly connected components and detecting communities, are tested. Additionally, local clustering coefficient and single-source shortest paths are part of the benchmark [IHN+16, p. 1317]. Furthermore, the authors introduce and consider so-called "choke points", which aim to reveal technological challenges in the implementation by getting knowledge of external experts [CHI+15a, p. 2]. For instance, excessive network utilization, large graph memory footprint, poor access locality, and skewed execution intensity [CHI+15a, p. 2].

Metrics The reported metrics are Edges per Second (EPS) as well as Edges and Vertices per Second (EVPS), which are defined as the ratio of the number of edges to the processing time and the ratio of edges and vertices to processing time, respectively [IHN+16, p. 1320]. In this case, processing time represents the time to execute an algorithm.

Technologies Initially, the benchmark provided implementations for the following platforms: Hadoop MapReduce, Giraph, GraphX, and Neo4j [CHI+15a, p. 4]. In [IHN+16], the benchmark was applied to the three community-driven platforms Giraph, GraphX, and PowerGraph, as well as to the three industry-driven platforms GraphMat, OpenG, and PGX [IHN+16, p. 1322].

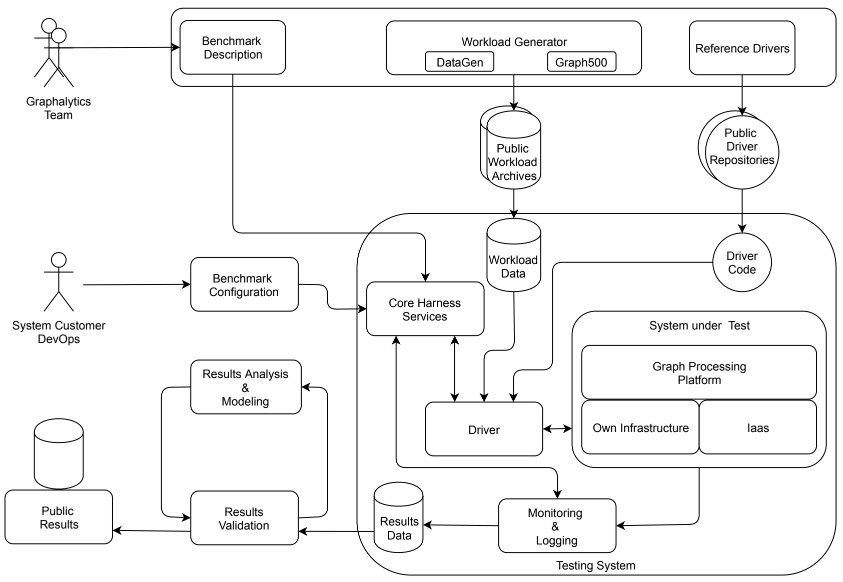


Figure 3.4: Graphalytics architecture and execution process, based on [IHN+16].

Design The benchmark comprises four steps: (1) graph selection, (2) platform configuration, (3) workload selection, and (4) benchmark execution [CHI+15a, p. 4]. Its implementation includes a workload generator, a benchmark configuration, a harness service, a driver, a SUT, a monitoring and logging component, as well as an analysis component [IHN+16, p. 1321]. The interplay among the components is illustrated in Figure 3.4.

Code The benchmark is publicly available as an open-source project[1]. In doing so, the authors also aim to achieve high code quality through, e.g., peer reviews, code analysis, and bug tracking [CHI+15a, p. 6]. To assure sustainability of the benchmark, it is developed with the purpose of integrating new data sets, algorithms, and platforms easily [CHI+15a, p. 3].

3.5.4 BigDataBench

BigDataBench [WZL+14] is a benchmark suite tailored for internet services covering the application scenarios of search engines, social networks, and e-commerce. Additionally, a scientific version of BigDataBench, called BigDataBench-S [TDD+17], is available, which covers common scenarios in physics, astronomy, and genomics. Another version of BigDataBench is BigDataBench-MT [HZS+15], where MT stands for multi-tenancy. This benchmark aims to assess the performance of a Big Data system, a data center, in particular, to handle a mix of scenarios. In this sense, the overall aim is to consider all aspects Big Data applications have to face nowadays, such as volume, velocity, variety, and veracity.

Data BigDataBench focuses on diverse data sources such as Wikipedia Entries, Amazon Movie Reviews, Google Web Graph, Facebook Social Network, e-commerce transaction data, and ProfSearch Person Resumes, which form representatives of text, graph, and table data. In addition, the data generator Big Data Generator Suite (BDGS) was developed in the context of the benchmark, which can generate synthetic data for various application domains [WZL+14, p. 6].

In BigDataBench-S, the used data sets are the ATLAS data set from high-energy physics as well as astronomical and genomic simulation data sets generated by SS-DB and GenBase [TDD+17, p. 1071]. Otherwise, the BigDataBench-MT is based on event logs from Sogou and cluster traces from Google [HZS+15, p. 5].

Workloads Version 5.0 of the BigDataBenchmark benchmark suite comprises micro, component, and application benchmarks. It covers search engines, social networks, and e-commerce from the internet services application domain [3]. Besides, the benchmark additionally supports recognition and medical sciences [3]. Depending on the application scenario and type, the workloads of the benchmark vary. The workload types are classified into AI, online services, offline and graph analytics, data warehouse, NoSQL, and streaming. Furthermore, micro benchmarks for basic datastore operations and relational queries are part of the suite. Overall, the suite contains 27 benchmarks and 13 data sources.

In BigDataBench-S, the workloads are classified into data manipulation queries and complex analysis for astronomy and genomics, as well as classification and regression for physics. Data manipulation queries contain selection, aggregation, and join operations, whereas complex analysis depends on the application domain. In astronomy, for instance, the complex analysis

[1]`https://github.com/ldbc/ldbc_graphalytics`

includes the intersection of images and sigma clipping. In genomics, the complex analysis comprises QR decomposition, SVD, and Covariance. On the other hand, BigDataBench-MT focuses on a mix of workloads considering long-running services and short-term data analysis jobs to simulate multi-tenancy support.

Metrics BigDataBench distinguishes between two types of metrics, namely, user-perceivable and architectural metrics [WZL$^+$14, p. 7]. User-perceivable metrics, on the one hand, consider the requests, operations, and data processed per second. On the other hand, Million Instructions per Second (MIPS) and Misses per Kilo Instructions (MPKI) are used as architectural metrics. The performance of the data analysis jobs is evaluated by their execution time, CPU usage, total memory size, Cycles per Instruction (CPI), and Memory Accesses per Instruction (MAI). Furthermore, the accuracy of the model predictions is measured [3].

Technologies The suite provides various implementations, for instance, Spark and Flink for offline and graph analytics or TensorFlow for AI [3]. Additionally, NoSQL systems are covered by implementations for MongoDB, and Hive [3]. For demonstration purposes, the authors use the Hadoop framework applying wordcount, naive Bayes, sorting, and page index for data analysis workloads.

In [TDD$^+$17], an implementation of BigDataBench-S for the genomics domain comparing Apache Spark, HiveOnMR, and HiveOnTez in combination with various data file formats is provided.

Design In BigDataBench-MT, the three-step execution process of the benchmark is supported by a user interface, which guides the user through the process of specifying the machines and workloads, selecting the benchmarking period, and finally generating the workload.

Code The benchmark software, the data sets, and the data generator are available under [2].

3.5.5 ShenZhen Transportation System (SZTS)

The benchmark aims to analyze the Smart Urban Transportation System of the Chinese city Shenzhen to gain useful insights into the future development of the infrastructure of the city [XYE$^+$16, p. 4340]. Although the city was recently founded in 1979, it has become one of the biggest cities in China, with approximately 18 million citizens and a surface area of 2,000 square kilometers [XYE$^+$16, p. 4340]. Shenzhen's transportation system comprises five subway lines approaching 118 stations supplemented by 936 buses with 10,300 bus stations and nearly 30,000 cabs.

Data The benchmark is based on four data sources, such as cellular phone records, cab Global Positioning System (GPS) records, smart-card transaction records, and cab deal information [XYE$^+$16, p. 4340], which are collected in real-time and stored in a data warehouse. For instance, cab deal information includes a timestamp, transaction account, distance, and start time [XYE+16, p. 4340]. Considering the data sizes, the bus stations, for instance, generate about 5 GB per day, whereas the cabs are responsible for 4.8 GB [XYE+16, p. 4342-4343].

Workloads In order to analyse the transportation system, the SZTS benchmark suite comprises five workloads, namely, *sztod*, *hotregion*, *mapmatching*, *hotspots* and *secsort* [XYE+16, p. 4342]. The first workload, called *sztod*, computes the movement of objects starting at A and heading to B considering a specific time interval. *Hotregion* computes the distribution of people and vehicles in a time interval. *Mapmatching* aims to compensate for measurement errors of GPS records. *Hotspots* identifies locations of high traffic, e.g., shopping centers or airports. And finally, *secsort* targets to sort incoming GPS records, which are received in the wrong order. Furthermore, the workloads are run with different input data sizes. For instance, *sztod* is executed with 20, 50, 100, 160, and 200 GB [XYE+16, p. 4349].

Metrics For the evaluation, the benchmark considers micro-architectural and job-level metrics. On the one hand, micro-architectural metrics are used to measure the performance of a specific node, e.g., Instructions per Cycle (IPC) or Last-level Cache Misses per Thousand Instructions (LLCMPKI) [XYE+16, p. 4345-4346]. On the other hand, job-level metrics are specific for the Hadoop characterization, e.g., Map Output/Input Ratio (MOI) or Time Map/Reduce Stage Ratio (TMRS) [XYE+16, p. 4343]. Furthermore, the benchmark analyzes the time consumption of OS and system libraries [XYE+16, p. 4344], for instance, Java-based libraries.

Technologies The benchmark basically targets the Hadoop framework and its analytical extensions Hive and Pig [XYE+16, p. 4341]. For instance, the *secsort* workload is implemented in Apache Pig [XYE+16, p. 4343].

Design The authors did not mention further information about how the benchmark was designed and implemented.

Code At the time of writing this thesis, only the documentation is still available as a download on [18]. A download of the benchmark, as well as the basic and large data sets, was not possible.

3.5.6 SparkBench

SparkBench is a technology-dependent benchmark aiming to support the optimization process of analytical workloads [LTW+17, p. 2575]. The growing popularity of the framework gives the relevance of the benchmark.

Data SparkBench utilities various data sets, such as Wikipedia, Amazon Movie Review, Google Web Graph, E-commerce, and Twitter [LTW+17, p. 2577]. Considering the data sets used by the benchmark basically different data types, structured, semi-structured, and unstructured, are covered [LTW+17, p. 2577]. To generate large data sizes, SparkBench additionally uses the Big Data Generator Suite (BDGS) [LTW+17, p. 2577]. In doing so, the benchmark covers the main characteristics of Big Data, namely, volume, velocity, and variety [LTW$^+$17, p. 2577].

Workloads SparkBench covers the following application types: ML, graph computation, SQL engine, and streaming applications [LTW$^+$17, p. 2577].

In the case of ML, the workloads comprise Logistic Regression (LR), Support Vector Machine (SVM) and Matrix Factorization (MF) [LTW$^+$17, p. 2577]. LR and SVM are tested with the Wikipedia data set, and MF with the Amazon Movie Review data set [LTW$^+$17, p. 2577].

Graph computation considers algorithms such as PageRank, SVD++ and TriangleCount [LTW$^+$17, p. 2578]. Amazon Movie Reviews are used as input data for SVD++ and Triangle-Count, and the Google Web Graph data to test PageRank [LTW$^+$17, p. 2578].

The SQL engine of Apache Spark is tested by select, aggregate, and join query workloads with Hive on Spark and Sparks native SQL capabilities [LTW$^+$17, p. 2578]. In this context, the queries target a database representing product orders on an e-commerce website [LTW$^+$17, p. 2578].

The streaming application feature of Apache Spark is evaluated by two use cases [LTW$^+$17, p. 2578]. The first one determines the most popular tweets on Twitter in the last 60 seconds. The second use case, called PageView, determines statistics such as counts of active users and pages on synthetically generated user interactions.

Furthermore, the workloads are executed with different parameter configurations of Apache Spark. This includes the configuration of the RDD cache size, parallelism, executor, and data compression [LTW$^+$17, p. 2583-2586].

Metrics During the execution of the benchmark, SparkBench measures various system resources such as CPU, memory, disk, and network usage. In this regard, SparkBench monitors the job execution time, the data process rate, shuffle data size, the input and output data sizes, and the average resource consumption [LTW$^+$17, p. 2578]. Nevertheless, the authors consider the job execution time as the most valuable metric to measure an improvement in performance due to optimizations. To measure the accuracy of the generated models, the RMSE is used [LTW$^+$17, p. 2580].

Technologies SparkBench is specifically tailored to benchmark Apache Spark and its previously discussed ML features, graph computation, streaming, and SQL processing.

Design The design of SparkBench is based on the following components: a data generator, workloads, workload suites, and spark-submit configurations [15]. A benchmark is executed as a spark job based on a nested configuration of spark-submit configurations and worksuits, which comprise a set of workloads. The data to run the benchmark has to be provided before the execution of the benchmark starts. The benchmark provides two data generators, one for k-Means and the other for LR. Own data generators can be integrated easily by implementing the appropriate interfaces, as explained in [16].

Code The implementation of SparkBench is publicly available under the Apache-2.0 License as an open-source project on GitHub [28].

3.5.7 StreamBench

In contrast to SparkBench, StreamBench [LWXH14] is not limited to benchmark only one specific streaming processing engine but multiple, namely, Apache Spark and Apache Storm. Nevertheless, StreamBench has to be classified as a technology-specific benchmark.

Data In order to represent a real-world scenario, the benchmark uses the AOL Search data and the CAIDA Anonymized Internet Traces Dataset.

Workloads StreamBench encompasses four benchmark suites: a performance, a multi-recipient performance, a fault tolerance, and a durability workload suite. Each suite assesses different aspects of the stream processing engines and therefore contains a different subset of available workloads or parameter settings. In general, StreamBench specifies three target aspects of stream processing engines, which are covered by the seven workloads.

The first aspect relates to processing numeric and textual data types. The second aspect considers the complexity of computations, such as sample, project and filter data. And the last aspect aims to test the ability of the processing engine to combine stored and streamed data. The seven workloads of the benchmark are *identity*, *sample*, *projection*, *grep*, *word count*, *distinct count* and *statistics*. *Identity*, *sample*, *projection* and *grep* are simple workloads which are applied to textual data. *Identity* acts as a no-operation (no-op) to provide a baseline for the other workloads. The *sample* workload extracts data with a specified probability from the incoming data stream. *Projection* represents the extraction of particular fields and *grep* checks the data for specific content. The workloads *word count*, *distinct count* and *statistics* are more complex since they are more demanding in the number of computational steps. *Word count*, for instance, is known as the hello-world example for many Big Data applications. It counts the frequency of words within a text. On the other hand, *distinct count* determines the number of distinct appearances of specific values. The only workload covering numeric data types is *statistics*, which determines the minimum, maximum, sum, and average for an attribute in the stream.

The performance, multi-recipient, and fault tolerance workload suites contain all seven benchmarks and only differ in the applied data scale sizes and the number of recipients in the cluster. In contrast, the durability workload suite only contains the *word count* benchmark with two available data scale sizes to test the availability of the processing engines within a time interval of two days.

Metrics In the case of pure performance, the determined metrics are throughput and latency. For the fault tolerance benchmark, two additional metrics, the Throughput Penalty Factor (TPF) and the Latency Penalty Factor (LPF), are introduced. Both metrics represent a relative factor comparing the throughput and latency in a cluster for the case in which all nodes are available to the case of a drop out of a single node in the cluster.

Technologies The benchmarked technologies are Apache Spark and Apache Storm.

Design The authors state that the design of StreamBench allows to benchmark any stream processing framework since it is decoupled from the implementation. More precisely, their design aims to decouple data generation from data consumption. In order to do so, they integrated a message system between the data generation cluster and data streaming cluster [LWXH14, p. 73]. For this purpose, they used the message system Apache Kafka in their experiments.

Code At the time of writing, no source code of the benchmark was available, but it was planned to publish it as an open-source project on GitHub [LWXH14, p. 78].

3.5.8 Yahoo! Cloud Serving Benchmark (YCSB)

YCSB was developed to compare cloud data serving systems, as the authors call it in their paper [CST+10, p. 143]. More precisely, the benchmark focuses on the evaluation of cloud OLTP applications, which not necessarily support ACID-transactions but rather BASE-transactions, such as NoSQL databases [CST+10, p. 143].

Data The data used during the execution of the benchmark is generated synthetically by YCSB. In the process, the underlying data model represents a row in a database with a predefined number of fields. Furthermore, the size of the fields can be configured. For instance, by default a primary key and ten additional fields, numbered consecutively from *field0* to *field9*, are generated containing 100 bytes of random ASCII characters [CST+10, p. 146]. The primary key is represented as a concatenation of the string “user” and a unique identifier, e.g., “user123456”.

Workloads By default, the YCSB provides seven basic workloads, namely, *workloada* to *wordloadf* and *tsworkloada* in the form of simple key value files. The included workloads aim to cover different application aspects such as persisting an action of a web session in *workloada* or adding and reading tags in *workloadb* [CST+10, p. 146]. For this purpose, each workload contains values to describe the proportion of read, update, scan and insert operations. Additionally, the number of records and operations as well as the distribution of requests, e.g., Zipfian, are basic parameters of each workload. For instance, *workloada* specifies a ratio of 50 percent read and 50 percent update operations based on 1,000 records and 1,000 operations. Furthermore, the number of fields and their corresponding size are part of the workload specification.

Metrics The benchmark measures the latency and throughput of the read, update, insert, and delete operations. The throughput is measured in operations per second and the latency in milliseconds.

Technologies Initially, the authors' motivation to develop the benchmark was to compare their system, called PNUTS, with Cassandra, HBase, and MySQL [CST+10, p. 144]. Since then, YCSB has evolved and currently supports a wide range of famous database systems, e.g., MongoDB, Cassandra, or Redis. A comprehensive and up-to-date list of supported database systems is given by the code repository in [24].

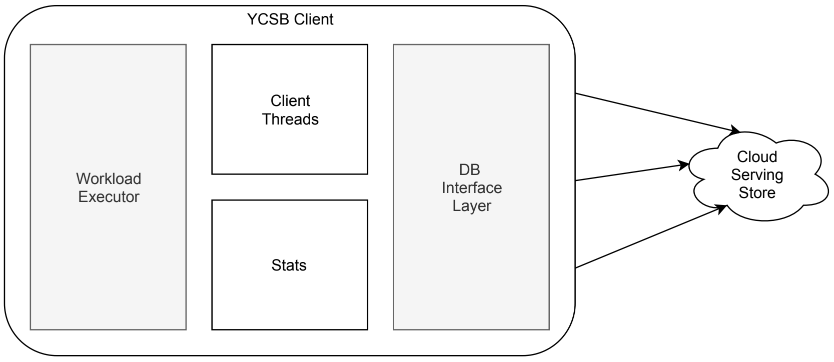


Figure 3.5: YCSB client architecture, based on [CST+10].

Design As illustrated in Figure 3.5, the main component of the benchmark is the YCSB client. The YCSB client is responsible for generating the data and for running the workload against a specified database [CST+10, p. 148]. Threads represent users running operations on the database. Finally, the stats component collects and aggregates the results of the threads to generate the overall result of the benchmark. To extend the YCSB benchmark to support a new database system, a connector has to be developed. This includes implementing a read, an insert, an update, a delete, and a scan method of a Java-based interface [CST+10, p. 149]. Additionally, it is possible to create new workload executors.

The execution of the benchmark is divided into six steps [37]. First, the database to test needs to be setup, e.g., creating the appropriate database schema. In the second step, the connector for the target database has to be chosen. Third, the workload has to be selected. Then, some runtime parameters, e.g., the number of clients, have to be defined. Afterward, the data is loaded into the database. And finally, the benchmark is executed.

Code The benchmark is publicly available as an open-source project under the Apache-2.0 License [24]. The author of the thesis contributed to the YCSB project by its implementation of a PostgreSQL database connector [40]. More precisely, the implemented connector enables to benchmark PostgreSQL regarding its NoSQL support, e.g., storing JSON.

3.5.9 MLPerf

MLPerf is a benchmark suite made for Machine Learning (ML) and Deep Learning (DL) in particular with the goals to enable a fair comparison, to accelerate ML development, to ensure reproducibility and to minimize the benchmark effort in order to be beneficial for academia and industry alike [MTW+20, p. 2].

The benchmark is motivated by the specific characteristics of DL computations, which allow versatile statistic, hardware, and software optimizations [MTW$^+$20, p. 1]. For instance, the underlying hardware and its configuration affect the hyperparameter settings and thereby the results. Another challenge to tackle is that statistical approaches may lead to various correct results between multiple runs [MTW$^+$20, p. 1].

The benchmark is divided into a training benchmarking [MTW$^+$20] and an inference benchmark [RCK$^+$20]. As the name implies, the training benchmark targets to assess the performance focussing on training a model to achieve a certain accuracy, whereas the inference benchmark evaluates the latency and throughput of the trained models.

Workloads and Data (Training)
The workloads of MLPerf aim to cover typical ML tasks. These include image classification, object detection, instance segmentation, translation, recommendation, and reinforcement learning [MTW$^+$20, p. 5].

Since the ML tasks of the benchmark suite differ, each workload has its own related data set [MTW$^+$20, p. 5]. Image classification is based on ImageNet. Object detection and instance segmentation utilize COCO 2017. The translation workloads use the WMT17 EN-DE data set. Recommendation exploits the MovieLens-20M data set. Besides, a synthetic data generator is developed to produce large data sets that are based on characteristics of the initial data. And finally, reinforcement learning works with a Go ($9x9$) board.

Workloads and Data (Inference)
The workloads of the inference benchmark cover four scenarios, namely, single-stream, multi-stream, server, and offline [RCK$^+$20, p. 5-6]. Single stream considers the responsiveness of an application on a mobile device receiving one query at a time, whereas multi-stream covers handling a stream of queries such as autonomous vehicles do [RCK$^+$20]. Server targets to mimic online applications, which have to respond to a user interaction quickly. In contrast, offline evaluates the performance of batch-processing applications.

Metrics
In contrast to the previously discussed benchmarks, MLPerf's performance metric considers the time to train a model until a certain quality is reached [MTW$^+$20, p. 6]. In this context, the time to train includes all operations, which use the data. Nevertheless, system initialization, model creation, and initialization, as well as data reformatting, are excluded from the timing [MTW$^+$20, p. 6-7]. Due to the variety of workloads, the quality metrics are different. For instance, recommendation uses the Hit Rate (HR) within the top 10 items (HR@10) as an accuracy measure, whereas object detection applies mAP. To achieve verifiable results, the workloads have to be executed multiple times.

The inference benchmark of MLPerf applies different metrics for the four scenarios [RCK$^+$20, p. 5]. In the single-stream scenario, the 90-th percentile latency is measured. The multi-stream scenario focuses on the number of streams, which are subject to latency bound. The server scenario considers the queries per second, which are subject to latency bound. And finally, the throughput is part of the offline scenario.

Technologies

The inference benchmark provides reference models, which are based on the formats of TensorFlow, PyTorch, and ONNX [RCK+20, p. 4].

Design

As already said, the benchmark is designed to consider training and inference. Since the training benchmark focuses on training and the inference benchmark assesses pre-trained models with a certain quality, it complements the former [RCK+20, p. 2]. An inference benchmark submission comprises the following components: a SUT, a load generator, a data set, and an accuracy script [RCK+20, p. 7], as depicted in Figure 3.6. The SUT has to be provided by the submitter. The load generator is responsible for loading the SUT and run the benchmark according to the configured scenario. Besides, it collects information, records queries and responses, and reports statistics [RCK+20, p. 8]. Furthermore, the benchmark supports an accuracy and a performance mode [RCK+20, p. 8]. In the accuracy mode, the complete data set is considered, whereas the performance mode considers only a subset. For extensibility, MLPerf provides an interface between the SUT and the load generator. The complete submission and review process is explained in [RCK+20, p. 9].

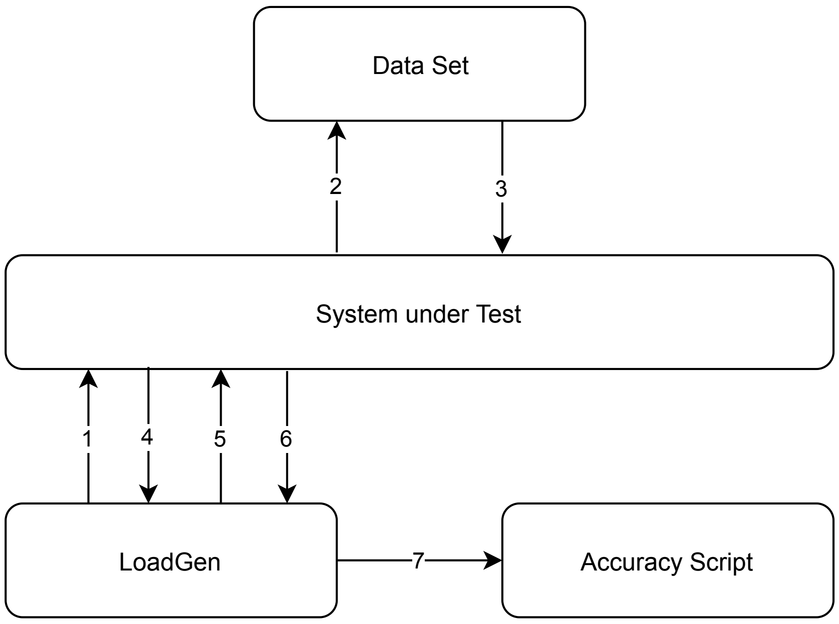


Figure 3.6: MLPerf Inference system, based on [RCK+20, p. 8].

Code

The source code of the benchmarks is publicly available as an open-source project under the Apache-2.0 License [5]. Most of the source code is implemented in Python, but C++ and shell scripts, are also used. For instance, the load generator is developed as a C++ application [RCK+20, p. 8]. Additionally, some Docker files are provided too.

3.5.10 Summary

Up to this point, several benchmarks have been presented, whose relevance in the context of recommender systems and for the remainder of the thesis follows.

TPC-DS and BigBench represent benchmarks that are based on a well-defined and elaborated data model of a retailer. Here, the data model comprises entities, which are essential parts of retail. In addition, the workloads cover representative tasks in retail. In doing so, both show how to develop standardized benchmarks covering a relevant application domain and representative tasks. Consequently, a benchmark for a recommender system has to consider and incorporate these findings.

Although not directly related to recommender systems, the SZTS benchmark also represents a benchmark motivated by a real-world scenario. It shows how to develop and implement a benchmark that measures metrics on different abstract levels and thereby leads to findings since they are not considered by the other benchmarks.

Considering Graphalytics, the benchmark reflects the increasing importance of analyzing social networks by graph-processing systems. In the context of recommender systems, especially workloads such as identifying weakly connected components enable to derive additional insights by considering the interest of related users. This leads to the finding that such relations have to be considered in a data model for recommender systems.

Recommender systems highly rely on up-to-date data to generate an accurate recommendation. Therefore, the underlying data storage systems have to perform basic database operations in a fast manner. First, to adopt the models based on the newest information but also to provide the recommendations to the user. In this sense, a benchmark for recommender systems has to take this into account, as shown by YCSB.

As data processing systems aim to process and analyze large data sets as well as streaming data, they are widely applied in recommender systems. Therefore, SparkBench and StreamBench are useful benchmarks to evaluate their performance, applying ML approaches, e.g., Matrix Factorization (MF), or dealing with an incoming stream of data generated by user interactions. Therefore, processing and streaming have to be considered in the implementation of recommender systems.

MLPerf provides insights about how to evaluate ML tasks by considering the trade-off between accuracy and performance, which are important aspects to consider in a benchmark for recommender systems. Especially, the separation of training and inference represents an interesting design approach.

An overview of the introduced benchmarks compared by type, application domain, technologies, and metrics is given in Table 3.1.

Table 3.1: Summary of the main aspects of the introduced benchmarks. At this point, the abbreviations have the following meanings: L (Latency), T (Throughput), Latency Penalty Factor (LPF), Throughput Penalty Factor (TPF), Edges per Second (EPS), Edges and Vertices per Second (EVPS), T2T (Time to Train), Instructions per Cycle (IPC), JET (Job Execution Time), Cycles per Instruction (CPI) and Million Instructions per Second (MIPS).

Benchmark	Type	Application Domain	Technologies	Metrics
TPC-DS	End-to-end	Decision Support	SQL-based	Query-based, e.g., QphDS
BigBench	End-to-end	Big Data Analytics	SQL-based Streaming-based Hadoop-based	Query-based, e.g., BBQpm
Graphalytics	System	Graph Analysis	Graph-processing	Graph-based, e.g., EPS, EVPS
BigDataBench	Suite	Internet Services Recognition Sciences Medical Sciences	Streaming-based ML-based NoSQL-based	Low-level, e.g., MIPS, CPI Quality, e.g., Accuracy
ShenZhen	End-to-end	Big Data Analytics	Hadoop-based	Low-level, e.g., IPC
SparkBench	Application	Machine Learning Graph Analysis SQL Streaming	Streaming-based	Low-level, e.g., JET Quality, e.g., Accuracy
StreamBench	Micro	Streaming	Streaming-based	Low-Level, e.g., L, LPF, T, TPF
YCSB	System	Storage	NoSQL-based	Low-level, e.g., L, T
MLPerf	Suite	Machine Learning	ML-based	Low-level, e.g., L, T, T2T Quality, e.g., Accuracy

3.6 Benchmark Model and Benchmark Execution Process

3.6.1 Benchmark Model

Based on the investigated benchmarks and their designs in Section 3.5, the components which comprise a benchmark implementation are deduced and explained.

System under Test
According to [Raa19], a SUT defines hardware, software, and connectivity components, which are part of the benchmark process but may also explicitly specify the ones which are not part of it. In the TPC-DS specification, a SUT is considered as “a collection of configured components used to complete the benchmark” [29, p. 66]. Depending on the considered benchmark objective and type, for instance, the hardware components are considered to be fixed to evaluate a specific software component. On the other hand, to benchmark different hardware components, the software component could be fixed. As the name implies, the connectivity components specify how the hardware and software communicate with each other. For instance, a benchmark could provide a connectivity component to communicate with the SUT. In these cases, it must be defined whether such communication is to be considered part of the SUT or not since it impacts the benchmark results.

Data Generator
In the first place, a data generator generates data used by a benchmark, for instance, loaded into the SUT. In addition, the data generator is responsible for stressing the SUT [BWT17, p. 15]. Such stress is commonly named workload. In [BWT17, p. 62], two types of workloads are mentioned, namely, synthetic and trace-based workloads. As the name implies, synthetic workloads generate data based on, e.g., a particular probability distribution, whereas trace-based workloads are based on a sequence of timed instructions [BWT17, p. 62]. Consequently, trace-based workloads, in contrast to synthetic workloads, are repeatable but are also easier to adapt to [BWT17]. Both approaches can be based on collected data of real applications. In this regard, the generation and ingestion of the data into the SUT might already be part of the stress test. For instance, generating a large data set to test a database. On the other hand, a data generator might stress the SUT on the already imported and generated data by invoking, for instance, SQL statements.

Benchmark Driver
Besides the benchmark driver, another term used by the authors in [BWT17, p. 15] is the experiment control component. Furthermore, sometimes the term harness is used, for instance, in [IHN$^+$16, p. 1321]. Nevertheless, neglecting the naming, its purpose is the same. In [CGL$^+$16, p. 28], the authors define the benchmark driver’s task to orchestrate the workflow during the benchmark execution on the SUT. The experiment control component mentioned in [BWT17] aims to manage the execution of the experiments. In this sense, the benchmark driver assures the interplay between the data generator and the SUT given a specific workload and configuration as specified in TPCx-BB. Furthermore, the benchmark driver is responsible for ensuring specific prerequisites of the SUT before workloads are executed.

Interface
To ensure smooth communication between the benchmark driver and the SUT, most benchmarks require the implementation of a specified interface. In this way, the benchmark driver and the SUT are decoupled. This enables a benchmark to support various SUT by implementing a corresponding interface or driver. For instance, in [CST$^+$10], the benchmark requires to implement a database interface layer to support new databases. Another example is the so-called driver implementation in [IHN$^+$16], which aims to integrate new platforms in the future.

Measurement Data Component
The primary task of the measurement data component [BWT17, p. 15] or logging component [IHN$^+$16, p. 1321] is to collect all necessary results of the benchmark run for further analysis. Accordingly, such a data component might be a database or a widely used file format, which provides a standard interface to answer queries [BWT17, p. 15].

Monitoring
Monitoring does not necessarily have to be part of a benchmark model, but it provides the benefit to identify bottlenecks in the SUT, the load generator, or the measurement process [BWT17, p. 15]. For instance, monitoring prevents to run long-running benchmarks with wrong parameter settings or configurations.

Configuration Files
Configuration files are used to configure the components involved in the benchmarking process. This includes setting parameters of the SUT and the benchmark, as well as selecting and configuring the workload to run.

3.6.2 Benchmark Execution Process

In this sense, the benchmark execution process, as illustrated in Figure 3.7, generally comprises the following activities:

Preliminary Activities
Before the benchmark can be executed, some preliminary activities have to be done. Depending on the benchmark and the targeted SUT, it might be required to implement an interface for the communication between the benchmark driver and the SUT. An example is to implement database-specific query operations, as explained in the YCSB benchmark in Section 3.5.8. This optional activity is represented as *Implement Interface* in Figure 3.7.

In the activity *Setup SUT*, the SUT is defined and setup correctly. For instance, this includes selecting hardware resources, software components, or even algorithms depending on the benchmark type. Furthermore, this activity involves the deployment of the software components or algorithms on the hardware resources. Next, the SUT must be configured appropriately by a given configuration, which serves for documentation purposes. The *Configure SUT* activity includes configuring software components such as processing and database systems. In the former case, this activity might involve configuring the appropriate data sources or structures. In the latter case, it might be required to create users and roles to access the database or generate a database schema matching the benchmark requirements.

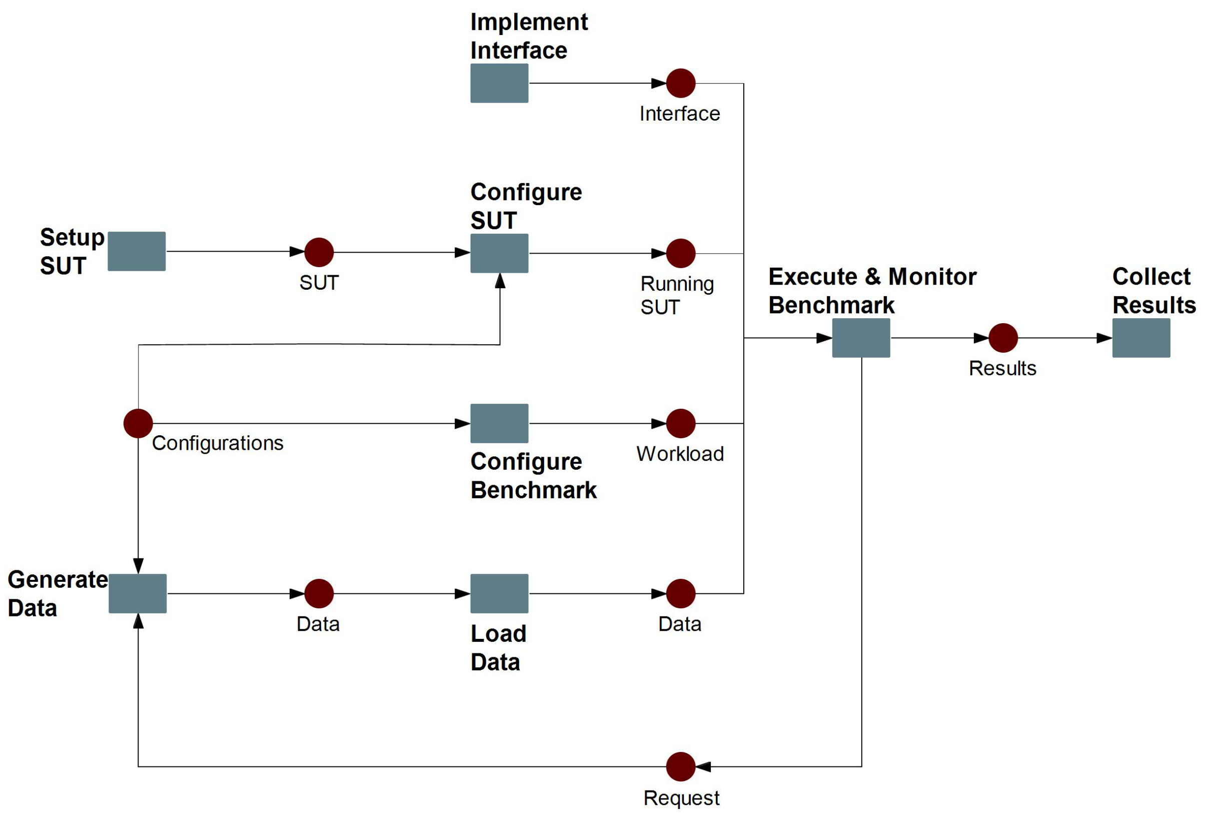


Figure 3.7: Illustration of the benchmark execution process and the involved activities.

Benchmark Activities

After performing the preliminary activities, the SUT can be benchmarked. Part of the activity *Configure Benchmark* is to specify the workloads and their corresponding parameters for the benchmark execution. For instance, a specific workload can be configured to mimic create, read, update, and delete operations by only one or multiple users simultaneously. Furthermore, the workload might support various data sizes and data types.

Besides, the activity *Generate Data* involves generating data representing the application scenario covered by the benchmark. For instance, generate data to mimic a retailer as BigBench in Section 3.5.2. This activity might be optional for some benchmarks since a data set is available for its execution, which is mostly not the case. After the data has been generated, the activity *Load Data* takes care to load it into the SUT. In some benchmarks, this step is already considered as one part of the benchmark. For instance, it might be of interest to measure how long it takes for a specific database to load it.

Now that the interface, the SUT, the workloads, and the data have been generated and configured appropriately, the activity *Execute & Monitor Benchmark* can be executed. At this point, no changes to the active components are allowed, which affect the benchmark. In the following, the benchmark driver orchestrates to stress the SUT with workloads or additionally requests new data from a data generator, as illustrated by *Request* in Figure 3.7. This additionally reflects workloads based on streaming data. Furthermore, the current status of the SUT is monitored, for instance, to show CPU usage and memory consumption during the execution.

Finally, the benchmark results are collected in a proper format for further analysis represented in the activity *Collect Results*. Additionally, the SUT and the benchmark configurations should be part of the results too. In doing so, external parties can reproduce the complete benchmark process.

4 A Benchmark Concept for Recommender Systems based on Omni-Channel Data

This chapter develops a benchmark concept for a recommender system benchmark based on omni-channel data. The idea of the benchmark concept is to show how to evaluate a recommender system that utilizes data from multiple channels.

In this regard, the number, as well as the characteristics of each channel, are depending on the application domain of the recommender system. For instance, in e-commerce, such channels include a social media account, an email service, or an online shop. In contrast, in e-library, only an online service and newsletters are used to retrieve user feedback. Furthermore, the collected information within a channel differs among various application domains. In online retailing, for example, users may use social media heavily to engage with their preferred retailer by likes and comments, whereas in the scientific context, users instead tend to use newsletters to indicate interest.

Nevertheless, the information might be useful to generate more accurate recommendations by incorporating this implicit feedback in the recommendation process. Since most of the collected data only indicates interest in the form of implicit feedback, the impact of the channels has to be considered in the proposed benchmark concept. In this sense, the data among the various channels can be combined, considering different aggregations to apply recommender system approaches afterward. In doing so, the proposed benchmark concept complements and generalizes current evaluations of recommender systems, which mainly focus on one signal type.

Finally, the results have to be evaluated, considering business, user, and technical aspects covered by different metrics. In this sense, the proposed concept represents an end-to-end benchmark. In the following, a benchmark concept and its components are described.

4.1 Channels and Signal Types

Nowadays, companies are able to reach their user in manifold ways to distribute, promote, and sell their products and services. In marketing and distribution, the term channel is commonly used to refer to this concept. According to [49] and [Emr08, p. 209], examples of channels are stationary, online, mobile, call center (phone), social media, catalogs, email, websites (online service) and personal contact. In [Hol14], specifically, online marketing channels are investigated intensively. Furthermore, channels can be distinguished by whether the information is pushed to or pulled by a user [Emr08, p. 35]. Depending on the provided number

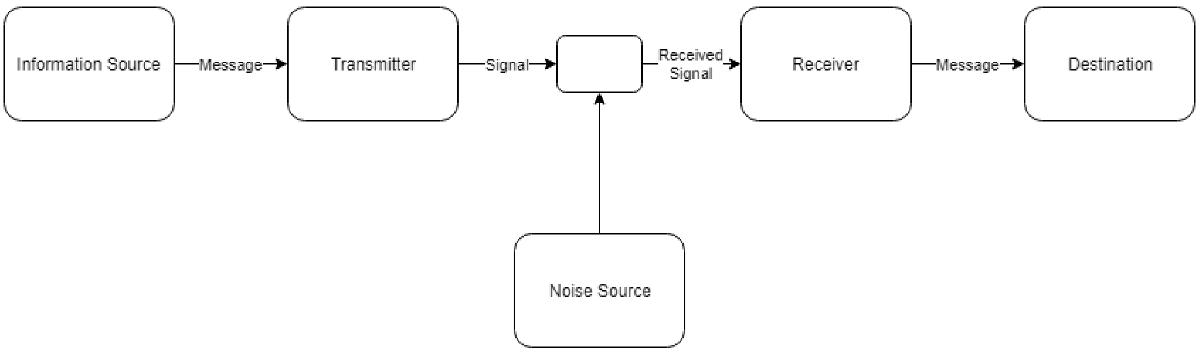


Figure 4.1: The figure illustrates the generic communication model of Shannon and Weaver, based on [Sha48].

of channels, their integration, and interplay, they can be categorized into omni-channel, multi-channel, and cross-channel management, as defined in [49]. In this context, omni-channel management is characterized by a seamless interplay and the highest integration among the different channels, whereas multi-channel considers them mostly separately or only partially connected. Cross-channel management, on the other hand, is considered as somewhere in between. By their interactions with the channels, users provide useful information about their behavior and interests in services. To cover different kinds of interactions, the broader term signal is used in the following, which was introduced by Shannon and Weaver in their mathematical communication model [Sha48] and is used in the communication research area [Emr08, p. 56-59]. An illustration of the communication model is depicted in Figure 4.1.

Taking the phone as an example, a user is able to request information about a particular item by calling a call center, for instance, its release date and so indicating interest for it [Hol14, p. 271]. A user can also write an email to a company's customer service to complain about or praise the quality of an item [Hol14, p. 271]. On the other hand, a company can send information about upcoming items in the form of a newsletter. Here, a signal type could represent a newsletter's delivery, and other signals might represent whether a user opens or clicks it. Considering social media, signals appear in the form of comments, likes, and shares [Guy15], [Hol14, p. 189]. In this regard, a company can collect this information to infer interest, for instance, by analyzing the content of a comment regarding an item. In the case of an online service for clothes, a user's behavior can be represented by signal types such as clicks, searches, reviews, or purchases. Other examples are clicks, purchases, ratings, and reviews, as mentioned in [WM18]. In [AB15] and [53], Netflix uses the term signal to refer to recent information collected by their services, for instance, what movie a user watched or how it was rated. Reading, sharing, printing, and commenting on an article are examples of signals generated regarding a news service [KJJ18]. Although the previous examples are related to signal types generated digitally, users are still using personal contact or brick and mortar stores to get information about an item in a personal conversation or buy it directly. At this point, for instance, loyalty cards provide a way for companies to link purchase information to individual users.

Besides, signals are characterized by their communication direction. Depending on whether one, two, or more participants are part of the communication, it is referred to as uni, bi, or

Table 4.1: Overview of various channels of a fictitious retailer, their corresponding signal types and communication direction.

Channel	Signal type	Direction
Phone	Request product information	bidirectional
	Complain about a product	bidirectional
	Praise a product	bidirectional
Email	Send a newsletter	unidirectional
	Open a newsletter	unidirectional
	Click a newsletter	unidirectional
	Complain about a product	bidirectional
	Praise a product	bidirectional
	Request product information	bidirectional
	Send product information	unidirectional
Social media	Comment	multidirectional
	Like	multidirectional
	Share	multidirectional
Online service	Click	unidirectional
	Search	unidirectional
	Review	multidirectional
	Purchase	unidirectional
	View	unidirectional
	Rate	unidirectional
Personal contact	Buy item	bidirectional
	Request information	bidirectional

multidirectional. For instance, in social media, many users participate in the same conversation about products by comments or likes. The same applies to reviews of products in an online service. On the other hand, rate, purchase, or click signals only require one participant.

These examples illustrate the importance of channels and signal types for modern recommender system implementations since they enable to draw conclusions about a user's interests by considering multiple perspectives. Table 4.1 summarizes channels, signal types and communication directions.

4.2 Data Model

The data utilized in the proposed benchmark concept should reflect the real-world application domain of recommender systems. Therefore, initially, a realistic Entity Relationship Model (ERM) has to be introduced and described, as depicted in Figure 4.2. The aim of this ERM is to identify all relevant entities and their relationships, which compose an omni-channel

Table 4.2: The table describes possible entity values for an exemplified movie streaming provider based on the ERM in Figure 4.2.

Entity	Description	Value
U	set of users	{A, B, C}
I	set of items	{Stars Wars, Back to the Future, Star Trek, Matrix}
Ch	set of channels	{online service}
S	set of signal types	{rate}
D	set of devices	{AppleTV, Personal Computer}

recommender system.

In the center of the data model are signals on channels between users and items. Each user generates signals of a specific type and channel which are associated with one or multiple items, as explained in Section 4.1. For instance, a purchase signal can be clearly assigned to one specific item, whereas a search signal can cover many different items due to ambiguity in the search result. In this case, every item of the search result generates an own entry. On the other hand, an item can be part of various signals, either since different users interacted with it or one user used multiple channels for interaction. Therefore, the relation between items and signals is represented as a many to many cardinality. Furthermore, a user generates signals by various devices he or she owns, such as a smartphone, a tablet, or a personal computer. Beyond that, the device type plays an essential role in presenting recommendations regarding their number and arrangement. For example, in general, the number of products shown to the user differs between a smartphone and a personal computer due to space limitations. Nevertheless, it is possible to generate signals without explicit usage of a device, e.g., buying in a brick and mortar store. Additionally, each signal is characterized by its type and temporal occurrence. Here, a signal type can be part of different signals, but a signal is of exactly one specific type. The time reflects the occurrence of a signal. Consequently, a signal occurs at a specific time, but multiple signals can occur at the same time. Supplementary, each signal is related to a channel. In addition, a signal has to contain content, for instance, a purchase, rating, or comment. Concluding, the data model has to consider all items provided as recommendations to a user.

To provide a generic data model, items and users are described by item and user features, respectively. In doing so, the data model is not limited to specific recommender system approaches. Item features describe item characteristics by their metadata, e.g., a description, color, or genre. Consequently, user features characterize a user by, for instance, its demographic information, buying behavior, or interests. Moreover, the data model is able to represent diverse application domains by specifying different channels and signal types.

In Table 4.2 an example of possible entity values for a movie streaming provider is represented. Here, the items represent movies, and the content of the signals is their corresponding rating represented on a numerical scale from one to five. The provided movies for the three users A, B, and C are Star Wars, Back to the Future, Star Trek, and Matrix. Furthermore, the users are using AppleTV and PCs to stream the movies.

Another example, based on a fictitious online retailer, illustrates the usage of multiple

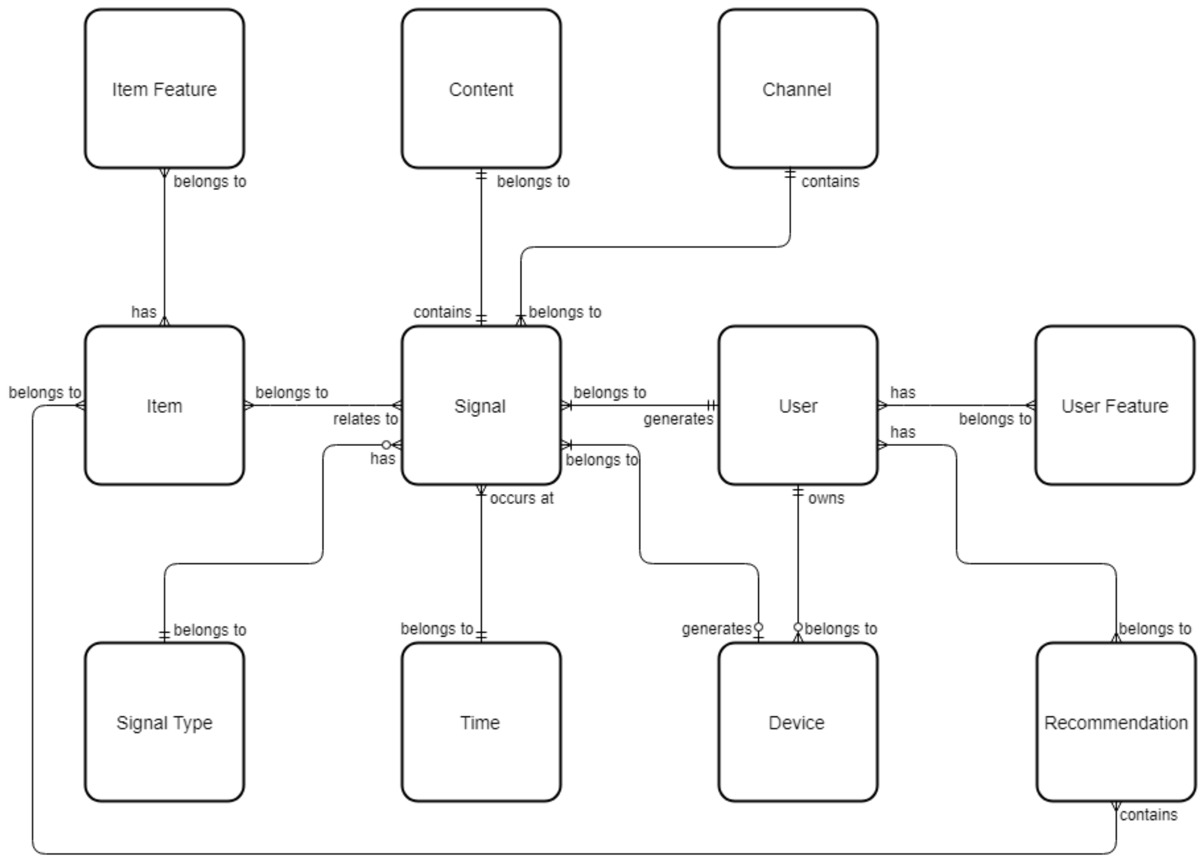


Figure 4.2: An ERM in Crow's foot notation. It comprises the entities and their cardinalities required for benchmarking omni-channel recommender systems.

Table 4.3: The table lists signals collected by a movie streaming provider. Here, the representation is based on the structure of a relational database.

ID	User	Item	Channel	Signal type	Content	Time	Device
1	A	Star Wars	online service	rate	5	2020-02-25	AppleTV
2	B	Star Trek	online service	rate	3	2020-02-26	PC
3	C	Matrix	online service	rate	4	2020-02-27	AppleTV

Table 4.4: The table describes possible entity values for an exemplified online retailer based on the ERM in Figure 4.2.

Entity	Description	Value
U	set of users	{A, B, C, D, E}
I	set of items	{t-shirt, trousers, jacket, pullover, socks}
Ch	set of channels	{online shop, social media}
S	set of signal types	{click, purchase, comment, view}
D	set of devices	{iPhone, Samsung, Huawei, PC, iMac}

channels and signal types. In this case, the items are products such as t-shirts, trousers, jackets, pullovers, socks, and signal types such as clicks, purchases, comments, or views collected over an online shop and social media. In this regard, the content of purchases and views is defined as a numerical value indicating their amount, whereas comments are represented as unstructured text data. Table 4.4 illustrates exemplary user, item, channel and signal values for the online retailer application domain.

Table 4.5: The table lists signals collected by an online retailer. At this point, the representation is based on the structure of a relational database.

ID	User	Item	Channel	Signal type	Content	Time	Device
1	A	t-shirt	online shop	purchase	1	2020-02-25	iPhone
2	A	trousers	online shop	purchase	2	2020-02-26	iPhone
3	B	t-shirt	online shop	purchase	3	2020-02-25	Huawei
4	B	trousers	online shop	purchase	3	2020-02-26	Huawei
5	B	jacket	online shop	purchase	1	2020-02-27	PC
6	C	trousers	online shop	purchase	3	2020-02-25	iPhone
7	C	jacket	online shop	purchase	1	2020-02-26	iMac
8	C	socks	online shop	purchase	5	2020-02-27	iMac
9	D	t-shirt	online shop	purchase	1	2020-02-25	Honor
10	D	trousers	online shop	purchase	2	2020-02-26	PC
11	D	pullover	online shop	purchase	3	2020-02-27	Honor
12	E	trousers	online shop	purchase	1	2020-02-25	Samsung
13	E	pullover	online shop	purchase	2	2020-02-26	PC
14	A	pullover	online shop	click	1	2020-02-25	iPhone
15	B	socks	online shop	view	4	2020-02-25	Samsung
16	C	t-shirt	social media	comment	Great t-shirt.	2020-02-27	PC
17	D	pullover	social media	comment	Bad quality.	2020-02-27	Honor

In the context of utilizing data to benchmark recommender systems, signals could represent rows within a table in a relational database, as illustrated by the Tables 4.3 and 4.5. In Table 4.3, user A, for instance, *rated Stars Wars* with *5* on *2020-02-25* using his or her *AppleTV* for streaming the movie. Additionally, Table 4.5 shows different signals generated by users during their interactions. For instance, user A *clicked* on a *pullover* on *2020-02-25* utilizing its iPhone, user C *commented* his or her experience about a *t-shirt* and user B *purchased three trousers* via his or her smartphone. Here, the tables are generated by connecting the different entities by their foreign keys. For instance, a user or item in a row could be represented by a unique identifier pointing to the corresponding table of the entity. The same applies to the entities' channel and signal type as well as content, time, and device.

Accordingly, a signal can be defined as a tuple $(id, u, i, ch, s, c, t, d)$. Here, id denominates a unique identifier to distinguish among the different user signals. Furthermore, u denotes a user of the set U of users, and i denotes an item of the set I of items. Additionally, s denotes a signal type of the set S of signal types on the channel ch of the set Ch of channels, and c defines the content provided by a signal. Finally, d denotes a device that generates a signal at time t.

4.3 Data

Besides a data model, appropriate data is of relevance for the benchmark execution. As elaborated in Section 3.5, benchmarks use public data sets or provide a data generator out of the box. Since both approaches should be supported by the benchmark concept, they are discussed in the following.

4.3.1 Own and Public Data

In case a benchmark user intends to use his or her own data, he or she is in charge of converting it into the data model introduced in Section 4.2 and illustrated in Figure 4.2. Since the internal representation of the data differs among application domains, a benchmark user has to implement a customized Extract, Transform, and Load (ETL) process to ingest the required data into the target data model.

To force a benchmark user to perform this task might sound like an additional burden and thereby violating the requirement on a benchmark to be simple (Section 3.4). However, this leads to two advantages. First, if the benchmark user decides to publish the data, its format is already standardized by the given data model, making it easier for third parties to use. As a consequence, the reproducibility and verifiability of the results are strengthened, as required by a good benchmark (Section 3.4). Second, this representation enables the benchmark to consider the task of a recommender system to load the appropriate data for the execution of a recommender system approach. This task will be explained in the further sections.

4.3.2 Data Generation

Although the general idea of the benchmark is to apply it to a company's own or publicly available data, it also aims to provide a way to generate synthetic data sets to mimic real-world data sets from different application domains. More specifically, the generated data has to reflect the amount and distribution realistically. The usage of synthetic data not only

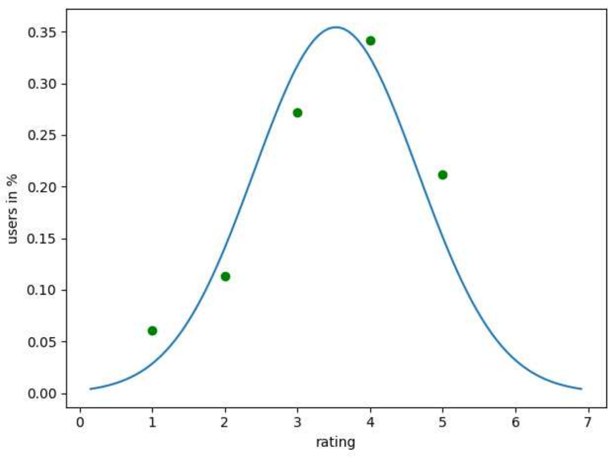


Figure 4.3: The green dots depict for each possible rating value (1 to 5) its occurrence within the MovieLens-100K data set scaled to 0 to 1. For instance, approximately 7 percent of the movies have a rating of 1 and 33 percent a rating of 4. The average rating in the data set is approximately 3.5, and the corresponding deviation is 1.1. The blue line depicts an approximated normal distribution with $\mu = 3.5$ and $\sigma = 1.1$.

enables to share the results of the benchmark but also the data itself. In doing so, a higher understanding and reproducibility are given by rerunning the benchmark, as discussed in Section 3.4. Figure 4.3 aims to convey the idea behind this.

Here, the publicly available MovieLens-100K data set is used. The data set comprises $100{,}000$ ratings from 943 users for $1{,}682$ movies on a scale from one to five. In this sense, the data represents rating signals, as illustrated in Table 4.3. Here, the five green dots depict the ratings (1-5) on the x-axis and the percentage of their occurrence within the data set on the y-axis scaled to $[0, .., 1]$. Considering the dot distribution, it seems to follow a bell-shaped curve with only a few users giving very low or very high ratings and a peak for medium ratings of 3 and 4. Therefore, a normal distribution can be used to approximate the dots as depicted by the blue line in Figure 4.3. The normal distribution depends on the two values μ and σ as shown in the corresponding equation:

$$f(x, \mu, \sigma) = \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma^2}} e^{\frac{(x-\mu)^2}{2 \cdot \sigma^2}},$$

where μ is the average movie rating and σ the standard deviation, which in this case, are 3.5 and 1.1, respectively.

The approximation serves to generate synthetic data sets. Additionally, since the data is generated by imitating real-world data sets, it is further desirable to create similar data on different scales. Considering recommender systems, each application domain differs in the number of provided and available channels. Furthermore, each channel is characterized by varying and diverse user behavior in the form of signals. Consequently, multiple aspects of the data distribution have to be considered for synthetically generated data.

User Aspects

First, the data has to reflect the user behavior regarding the generated signals within each channel individually. For instance, in online retailing, some users might prefer to write detailed reviews about items directly in the online store, whereas others engage with their favorite retailer by liking or commenting on items shared on social media. Consequently, the number of generated signals in the online store and social media differs depending on the target audience. Considering Amazon as one of the most popular online retailers, which provides a wide range of items, the amount of reviews generated by users on its platform exceeds the amount generated by a smaller online retailer, which may provide only niche products and does not act globally. In another application domain, the situation might be the other way around. According to this, the number of generated signals varies among different application domains and channels and has to be considered in the data generation process.

Second, the data has to reflect a realistic number of user signals with items as well as signals an item receives by users. More specifically, the generated data has to capture the behavior of rather conservative or passive users and frequently active ones. In this context, it might be of interest to specify a minimum number of user signals to enable certain approaches. For instance, the MovieLens-100K data set[1] provides at least 20 ratings per movie.

Third, the time of the occurrence of signals should mimic real-world user behavior. For instance, users might tend to engage with a service, preferably after work than in the morning. Furthermore, it might be the case that user signals increase on certain days, e.g., Christmas or Black Fridays.

Fourth, considering an individual user, the generated data has to reflect his or her behavior. More specifically, the data has to describe a reasonable amount of signals among the channels a user uses. In this sense, the data generation process has to respect different user profiles. For instance, one user profile might be characterized as a heavy online shopper, which additionally writes many reviews. Considering the consistency of the data, it could make sense for a user to like a movie on social media but giving it a low rating after streaming. The reason for that might be that he or she was looking forward to seeing it at the beginning but disappointed afterward. On the other hand, it does not make sense to rate a movie high and then dislike it. Another example is online retailing, where it is reasonable to write a critical review of a purchased item. User profiles could either be identified by applying classification approaches to known data sets or defined in advance since certain user behavior is assumed within the considered application domain.

In implementing these user aspects the benchmark covers the requirements *relevance* and *acceptance*, as elaborated in Section 3.4.

Item Aspects

Considering item aspects, the data has to take the popularity distribution of items for each channel in the application domain into account. In other words, items are not consumed equally since there are always items enjoying higher popularity than others. In doing so, the generated data reflects natural user behavior. For instance, in movie streaming, some movies have a generally higher demand since a famous actor is part of it or it was directed by a

[1]`https://grouplens.org/datasets/movielens`

reputable director. An exemplary popularity distribution is depicted in Figure 4.4a. Here, the data represents how many times a specific movie has been rated by users.

Furthermore, the data has to consider the overall average preferences given to the items. In Figure 4.4b, the average rating distribution of the MovieLens-100K data set is given on a scale of one to five. The distribution illustrates that most movies are rated higher than 1. In addition, Figures 4.4c and 4.4d illustrate that popular items do not have to be rated high and highly rated movies do not have to be popular. Another example is the implicit feedback expressed by users through the number of purchased items, increasing confidence in a particular signal.

Consequently, these item aspects further strengthen the requirements *relevance* and *acceptance*, as mentioned in Section 3.4.

Technical Aspects

In this case, the data generation has to consider the dynamic of user signals. For instance, a real-world recommender system constantly receives new user signals. The quicker each new signal can be incorporated into the recommender system, the more current the provided recommendations. Therefore, the data generation additionally has to be able to generate not only static or batch data sets but also streaming data sets.

Besides, the data distribution must be able to scale with a varying number of users and items, preserving the previously mentioned aspects. For instance, how does a doubling of users affect the final recommendation results regarding runtime and accuracy? The same has to be considered by varying the number of items.

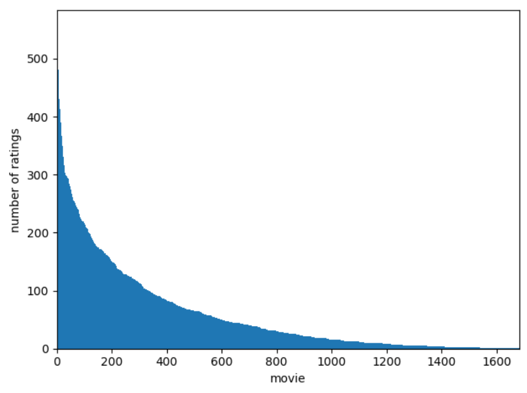
As a result, the implementation of these technical aspects leads to a scalable benchmark as required based on Section 3.4.

4.4 Data Processing

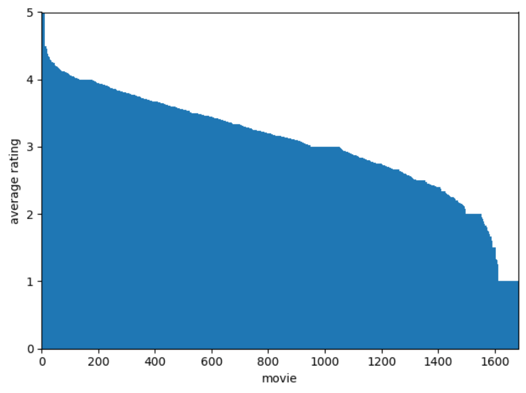
As discussed in Section 2.3 users provide implicit feedback by different signals, which occur in various representations. In this regard, three common preprocessing tasks are part of a recommender system and explained next.

4.4.1 User Matching

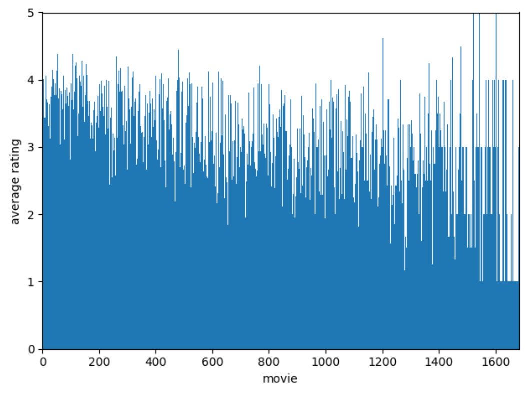
The first task aims to relate signals to users. Signals are rather useless if it is not possible to associate them with an individual user. It is relatively simple to assign purchases to customers when they use their accounts during online shopping. In the case of customers use a guest account, this task appears more complicated. At this point, technologies, such as cookies, help to connect the purchase with a possible customer. Another example is the delivery of newsletters. Customers may not provide an email address in their official account but use newsletters heavily. Here, a mapping of newsletter subscribers and customer accounts can be beneficial. The same applies to match names of customers on social media when they use their real names and the methods align with legal requirements.



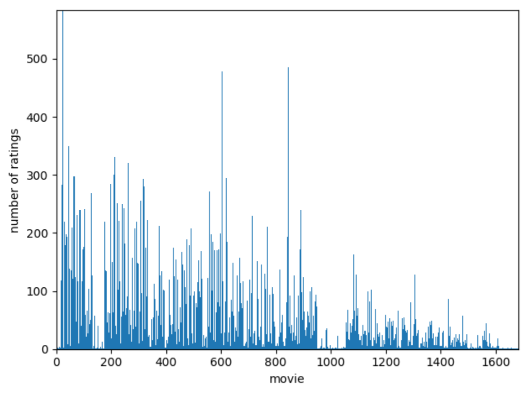
(a) The sorted movie popularity distribution. The figure illustrates that some movies witness significantly higher popularity than others.



(b) The sorted average movie rating distribution. The figure shows that most movies are rated higher than 1.



(c) The average movie rating distribution sorted by movie popularity. The figure illustrates that popularity does not necessarily correlate with a high rating.



(d) The movie popularity distribution sorted by their average ratings. The figure shows that highly rated movies do not have to be popular.

Figure 4.4: The subfigures represent different perspectives on the MovieLens-100K data set. Here, the values on the x-axis do not correspond to the values in the MovieLens data set.

4.4.2 Item Matching

The second task aims to relate user signals to items in case no direct relation is given. Such a situation appears, for instance, when a user searches for an item. In this case, the search result consists of multiple items covering the search term. Consequently, a user signal is derived for each search result and is considered as an indication of interest. Although these user signals might not have the same quality as those directly related to items, they provide an additional source of a user's interest or preference.

4.4.3 Content Matching

The third task aims to interpret the content of a given user signal. In this context, unstructured data, such as text, must be processed to classify its content as positive, negative, or neutral feedback. At this point, NLP from AI and ML are applicable. In particular, NLP approaches for sentiment analysis are of interest.

4.5 Data Aggregation

With the data at hand, the central idea is to aggregate the various signal types of users to investigate their influence on the recommendation results.

4.5.1 Binary Aggregation

The simplest form of aggregation is given by considering only one specific user signal type with explicit feedback, e.g., movie ratings, as illustrated in Table 4.3. However, in the following, a slightly more complex example, based on implicit user feedback of purchases from Table 4.5, is utilized for aggregation. In this case, each signal content contains the amount of a purchased item. In the process, the amount is neglected, and only the purchase itself is considered. More precisely, each signal content value larger than 0 is converted to 1, assuming an indication of interest. At this point, it must be noted that this assumption might not reflect the real interest of a user since a purchase does necessarily imply a positive user experience. The result of the aggregation is shown in Matrix 4.1. Using this aggregated data, exemplified product recommendations for user A are generated. Comparing user $A's$ similarity to the other users, he or she seems to share similar taste with users B and D since all bought t-shirts and trousers in the past. Consequently, it makes sense to recommend a jacket or a pullover to user A due to the behavior of user B or D, respectively.

$$\begin{array}{c} \\ \text{A} \\ \text{B} \\ \text{C} \\ \text{D} \\ \text{E} \end{array} \begin{array}{c} \begin{array}{ccccc} \text{t-shirt} & \text{trousers} & \text{jacket} & \text{pullover} & \text{socks} \end{array} \\ \begin{pmatrix} 1 & 1 & & & \\ 1 & 1 & 1 & & \\ & 1 & 1 & & 1 \\ 1 & 1 & & 1 & \\ & 1 & & 1 & \end{pmatrix} \end{array} \tag{4.1}$$

4.5.2 Equally Weighted Aggregation

However, since the idea is to aggregate multiple signal types of users, the purchase data is extended by the additional signal type comment from a social media channel, as illustrated in Table 4.5. In this case, the positive comment of user C from Table 4.5 regarding t-shirts is evaluated as an indication of interest and incorporated into Matrix 4.1 leading to Matrix 4.2. As a consequence, the recommendations for user A change. Now, besides user B and D, user A also shares a similar taste with user C, which results in an additional feasible recommendation of socks. At this point, the negative review of user D for the purchased pullover is not considered as indicated by the question mark.

$$\begin{array}{c|ccccc|} & \text{t-shirt} & \text{trousers} & \text{jacket} & \text{pullover} & \text{socks} \\ \text{A} & 1 & 1 & & & \\ \text{B} & 1 & 1 & 1 & & \\ \text{C} & \mathbf{1} & 1 & 1 & & 1 \\ \text{D} & 1 & 1 & & \mathbf{?} & \\ \text{E} & & 1 & & 1 & \end{array} \tag{4.2}$$

4.5.3 Weighted Aggregation

In the previous example, two aspects have been neglected so far. On the one hand, since no signal regarding user C and the t-shirt was available before and only one signal type was utilized for aggregation, the unknown value was simply added for this user-item combination. On the other hand, both signal types have been treated equally. This means it was assumed that the signal type purchase is as strong or weak as the signal type comment. Nevertheless, both aspects influence the recommendation results.

For instance, weightings of $\frac{2}{3}$ for the purchase and $\frac{1}{3}$ for the comment signal type produce Matrix 4.3. In this sense, purchase information is treated as more significant than comments. Considering the question mark in the Matrix 4.2 for user D and pullover, the weighted value is determined as $\frac{1}{3}$. The reason for that is based on weighting the purchase as a positive indication of interest with $1 \cdot \frac{2}{3}$ and the comment as a dislike with $-1 \cdot \frac{1}{3}$, which results in $\frac{1}{3}$. In this case, user A again shares a rather similar taste with users B and D as in the initial situation. But now, it is reasonable to recommend only a jacket to user A, due to the influence of the previous bad review of user D for the pullover.

$$\begin{array}{c|ccccc|} & \text{t-shirt} & \text{trousers} & \text{jacket} & \text{pullover} & \text{socks} \\ \text{A} & 2/3 & 2/3 & & & \\ \text{B} & 2/3 & 2/3 & 2/3 & & \\ \text{C} & \mathbf{1/3} & 2/3 & 2/3 & & 2/3 \\ \text{D} & 2/3 & 2/3 & & \mathbf{1/3} & \\ \text{E} & & 2/3 & & 2/3 & \end{array} \tag{4.3}$$

Generalized Definition

In the following, the applied aggregation approach will be generalized to Equation 4.4 and explained by the previous example.

$$w(u,i,Data,Ch^*,S^*) = \sum_{(u,i)\in Data, ch\in Ch^*, s\in S^*} \alpha_{ch,s} \cdot f_{ch,s}(u,i,c) \tag{4.4}$$

Here, the aggregated value for user u and item i on the data set $Data$ is denoted as $w(u,i,Data,Ch^*,S^*)$. The data set $Data$ comprises all available user signals as shown exemplified in Table 4.5. Additionally, the parameters Ch^* and S^* define the sets of considered channels and signal types. Furthermore, $\alpha_{ch,s}$ defines a weighting for each individual channel and signal type combination ch and s as a generalization of the previously applied weightings of $\frac{1}{3}$ and $\frac{2}{3}$. At this point,

$$\sum_{ch\in Ch} \sum_{s\in S} \alpha_{ch,s} = 1.$$

Additionally, a function $f_{ch,s}(u,i,c)$ is utilized to map the content c of a signal type s on a channel ch for a user-item combination to a numerical value, since the individual values are aggregated. In the following, two functions $f_{os,p}$ and $f_{sm,c}$ are applied to the channel online shop (os) and social media (sm) considering the signal types purchase (p) and comment (c):

$$f_{os,p}(\text{B,trousers,3}) = 1$$

$$f_{sm,c}(\text{C,t-shirt,“Great t-shirt.”}) = 1$$
$$f_{sm,c}(\text{D,pullover,“Bad quality.”}) = -1$$

Here, $f_{os,p}$, for instance, maps the number of purchased trousers of user B to 1, which, at least, represents an interest. On the other hand, the function $f_{sm,c}$ considers the comment of user C regarding a t-shirt and maps it to 1. The critical review of user D is mapped to -1. In the following, Equation 4.4 is applied to determine the weighted indication of interest for user C in the t-shirt concerning the previously mentioned channels and signal types.

$$\begin{aligned} w(\text{C,t-shirt,Data,\{os, sm\},\{p,c\}}) =& \alpha_{os,p} \cdot f_{os,p}(\text{C,t-shirt,NONE}) + \\ & \alpha_{sm,c} \cdot f_{sm,c}(\text{C,t-shirt,“Great t-shirt.”}) \\ =& \frac{2}{3} \cdot 0 + \frac{1}{3} \cdot 1 = \frac{1}{3} \end{aligned}$$

At this point, Ch^* contains the channel online shop and social media. Furthermore, the considered signal types S^* are purchase and comment. The weighting of the signal type purchase is specified by $\alpha_{os,p} = \frac{2}{3}$ and comment by $\alpha_{sm,c} = \frac{1}{3}$ on the corresponding channel. Since no purchase signal is available in this case, the content of NONE is mapped to 0 and the positive comment to $\frac{1}{3}$. This leads to the overall result of $\frac{1}{3}$.

4.5.4 Sequence-based Aggregation

Another possible aggregation approach could consider the time of occurrence of a user signal. Considering Matrix 4.2 again, the two conflicting signals of user D regarding the item pullover represent a good example. In this case, user D initially bought the pullover but afterward wrote a bad review about it. Applying an aggregation based on the weightings of the two signals still results in a positive indication of interest for the pullover, as shown in Matrix 4.3. However, it would make more sense to rely on the latest signal user D provided. Consequently, such a time-dependent aggregation leads to Matrix 4.5. As a result, a pullover is definitely not an appropriate recommendation for user A.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} \text{t-shirt} & \text{trousers} & \text{jacket} & \text{pullover} & \text{socks} \end{array} \\ \begin{pmatrix} 1 & 1 & & & \\ 1 & 1 & 1 & & \\ \mathbf{1} & 1 & 1 & & 1 \\ 1 & 1 & & -1 & \\ & 1 & & 1 & \end{pmatrix} \end{array} \tag{4.5}$$

In this sense, the aggregation function can be formulated as shown in Equation 4.6. Here, $latest$ defines the most recent signal for a user-item combination under consideration of all available channels and signal types.

$$w(u, i, Data, Ch^*, S^*) = \underset{\forall (u,i) \in Data, ch \in Ch^*, s \in S^*}{\textit{latest}} f_{ch,s}(u, i, c) \tag{4.6}$$

Beyond that, it is possible to combine a weighted and time-dependent aggregation function to weaken or strengthen the most recent user signals since, for instance, a review is considered to be more important than a purchase.

4.5.5 Generalized Aggregation

The data aggregation can be further generalized to Equation 4.7 to support multiple aggregation approaches. At this point, the aggregation function in Equation 4.4 is flexible in the sense that it is able to consider only subsets $Ch^* \subseteq Ch$ of the available channels or signal types $S^* \subseteq S$. To further extend its generality, in Equation 4.7 it is possible to only examine specific devices by specifying $D^* \subseteq D$. Furthermore, the aggregation function agg is capable to filter user signals which occur within a certain time interval defined by the input parameters t_{min} and t_{max}. This enables to exclude user signals from the aggregation, which, for instance, occur a long time ago and consequently are regarded as irrelevant.

$$w(u, i, Data, Ch^*, S^*, D^*, t_{min}, t_{max}) = \underset{\forall (u,i) \in Data, ch \in Ch^*, s \in S^*, d \in D^*, t \in [t_{min}, t_{max}]}{\textit{agg(u, i, ch, s, c, t, d)}} \tag{4.7}$$

4.6 Benchmarking Process

In Section 4.5, a derivation of recommendations based on different aggregation approaches is introduced to highlight their impact on the results. In this section, a systematic evaluation process to benchmark an omni-channel recommender system from a business, user, and technical perspective is introduced and explained sequentially.

4.6.1 Overview

Generally, the evaluation process of recommender systems comprises a training and testing step. Before an evaluation can be performed, it is important to define how to split the data, what recommender system approach(es) to apply, and what metric(s) to measure. Then, in the training step, a recommender system approach is applied to a specified train data set to generate a model. In this context, a model is a mathematical representation, which aims to characterize a given train data set. Subsequently, in the testing step, this model is utilized to check its generality on a test data set considering the defined accuracy metric(s), as discussed in Section 2.5.

Besides accuracy, metrics considering the runtime performance or memory consumption, as well as the response time and throughput of a recommender system, represent other important and valuable aspects within the evaluation process. In this regard, it is reasonable to regard the training and testing step separately to evaluate the effort to generate a model as well as the generation of the recommendations itself based on the model. This approach is also applied by MLPerf, as explained in Section 3.5.

Another aspect that is mostly neglected is considering the effort it takes to collect and load the required data to run a recommender system approach. Consequently, this benchmark comprises the scenarios of *data loading*, *model training* and *recommendation generation*.

4.6.2 Data Loading

Data loading considers the components and steps of a recommender system to retrieve the required data for its training. The faster a recommender system has access to the data, the earlier it can start its training and generate a model, as depicted in Figure 4.5.

Especially for an omni-channel recommender system, which incorporates multiple channels, it must consider whether the retrieval of additional user signals is worth the effort to request it. For instance, combining purchase and social media data might lead to better accuracy but also increases the retrieval time since more data has been requested.

However, this also applies to retrieving only one specific user signal. In this case, it might be more beneficial only to retrieve the last day, week, or month than the complete data since its influence on the accuracy is only minimal.

For this scenario, the *Response Time (T_{Res})*, as well as the *Memory Consumption (M_{Con})*, are of interest. T_{Res}, since it expresses the time between requesting and receiving the data. Also, M_{Con} gives information about the required hardware resources.

The following equations cover these metrics:

$$e_{T_{Res}} : Data \times \{s \in S\} \times \{ch \in Ch\}$$

$$e_{M_{Con}} : Data \times \{s \in S\} \times \{ch \in Ch\}$$

After the data has been loaded, a data aggregation approach is applied, and the training process of the recommender system starts.

4.6.3 Model Training

To demonstrate the recommender system evaluation process, Matrix 4.1 is extended to Matrix 4.8, denoted as $U_{os,p}$, by incorporating additional purchases (p) from the online shop (os). So now, in contrast to Matrix 4.1, it is known that user A bought a pullover and socks as indicated by the highlighted values in Matrix 4.8.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} \text{t-shirt} & \text{trousers} & \text{jacket} & \text{pullover} & \text{socks} \end{array} \\ \left(\begin{array}{ccccc} 1 & 1 & & \mathbf{1} & \mathbf{1} \\ 1 & 1 & 1 & & \\ & 1 & 1 & & 1 \\ 1 & 1 & & 1 & \\ & 1 & & 1 & \end{array}\right) \end{array} \tag{4.8}$$

In the first step, a data splitting strategy sp from a set of splitting strategies SP has to be chosen and applied to generate the previously mentioned train set and test set (Section 2.6.1). Here, the data set is split by considering the purchase date of the items, assuming that pullover and socks are the most recently purchased ones. The result are the train data set $U_{os,p}^{Train}$ and the test data set $U_{os,p}^{Test}$ as depicted by the Matrices 4.9 and 4.10. To illustrate the dimensions of a real-world test data set, the complete Matrix $U_{os,p}^{Test}$ is depicted, although it only contains the two purchases of user A.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} \text{t-shirt} & \text{trousers} & \text{jacket} & \text{pullover} & \text{socks} \end{array} \\ \left(\begin{array}{ccccc} 1 & 1 & & & \\ 1 & 1 & 1 & & \\ & 1 & 1 & & 1 \\ 1 & 1 & & 1 & \\ & 1 & & 1 & \end{array}\right) \end{array} \tag{4.9}$$

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} \text{t-shirt} & \text{trousers} & \text{jacket} & \text{pullover} & \text{socks} \end{array} \\ \left(\begin{array}{ccccc} & & & 1 & 1 \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{array}\right) \end{array} \tag{4.10}$$

In the next step, the train data set is used as input for a recommender system approach, e.g., user-user based Collaborative Filtering, short uu, as introduced and explained in Section 2.4.1. Since the algorithm is based on determining a neighborhood of similar users, a set of parameters P, e.g., the neighborhood size k and the minimum similarity sim among them, have to be specified. In this case, k is set to 2 and a user considered as a neighbor when a similarity of at least 0.49 is given, which leads to the following function:

$$uu : U_{os,p}^{Train} \times \{k = 2, sim = 0.49\} \mapsto \theta_{os,p}$$

The result of running the algorithm uu on the data $U_{os,p}^{Train}$ and parameter set P is the model

$\theta_{os,p}$. At this point, for instance, the CPU runtime to generate the model $\theta_{os,p}$ is measured in milliseconds on the given recommender system approach uu by applying:

$$e_{CPU} : uu \times U_{os,p}^{Train} \times \{k = 2, sim = 0.49\} \mapsto \mathbb{R}_{+}.$$

4.6.4 Model Testing

In the testing step, the model $\theta_{os,p}$ can be further evaluated under accuracy and performance aspects. Here, the runtime to generate the recommendations is determined by the following evaluation function:

$$e_{CPU} : \theta_{os,p} \times U_{os,p}^{Test} \times \{nrecs = 3\} \mapsto \mathbb{R}_{+}.$$

Here, in contrast to the first evaluation function, the model $\theta_{os,p}$, the test data set $U_{os,p}^{Test}$ and the number of recommendations $nrecs$ to generate are required to measure the CPU runtime, since they all have an impact on it.

Considering the accuracy, the model $\theta_{os,p}$ is evaluated based on the test data set $U_{os,p}^{Test}$ and a set M_a of accuracy metrics, for instance, normalized Discounted Cumulative Gain (nDCG) to measure the recommendation quality, as discussed in Section 2.5. In addition, the number of recommendations $nrecs$ to generate as a ranked list is set to 3. The following evaluation function e shows its application given the mentioned parameters:

$$e_{nDCG} : \theta_{os,p} \times U_{os,p}^{Test} \times \{nrecs = 3\} \mapsto \mathbb{R}_{+}.$$

The provided recommendations by the model $\theta_{os,p}$ based on algorithm uu for user A are jacket and pullover in this particular order. Unfortunately, no third product could be recommended, given the current configuration. Comparing the items within the ranked recommendation list to the items user A purchased results in an nDCG value of 0.5. In this context, a higher value implies a better recommendation result. The low nDCG value has two reasons. First, the user A is recommended a jacket, even though he or she did not buy one in the future. Second, the jacket is also ranked high in the recommendation list, decreasing the nDCG value.

Applying the recommender system approach uu on the aggregated data of the online shop and social media channel as illustrated in Matrix 4.2 leads to the following equation:

$$uu : A_{agg}^{Train} \times \{k = 2, sim = 0.49\} \mapsto \theta_{agg}$$

In this case, A_{agg}^{Train} denotes the aggregated train data set based on applying an aggregation function agg. In doing so, a model θ_{agg} is trained, which changes the ranked list of recommendations for user A to jacket, socks, and pullover. Consequently, the result of the evaluation function e based on the algorithm uu and the metric nDCG increases to 0.82. This is a significant improvement compared to the previous result of 0.5. Therefore, from the accuracy perspective, in this example, it makes sense to incorporate the social media channel data into the training step.

Taking the CPU runtime performance into account, no measurable difference could be recognized due to the small number of users and items in this example. However, this must not always be the case since the number of incorporated signals may increase the computational effort in determine neighborhoods for uu. This means that a higher accuracy may not be worth the high demand for computation.

To sum up, the goals of this example were to check whether the incorporation of the social media channel improves the recommendation results, based on the accuracy metric nDCG considering the CPU runtime during the train and test step. Having said that, the goals of the evaluation are defined in the following mathematical representations:

$$e_{CPU} : uu \times A_{agg}^{Train} \times \{k = 2, sim = 0.49\} \approx e_{CPU} : uu \times U_{os,p}^{Train} \times \{k = 2, sim = 0.49\}$$

$$e_{CPU} : \theta_{agg} \times A_{agg}^{Test} \times \{nrecs = 3\} \approx e_{CPU} : \theta_{os,p} \times U_{os,p}^{Test} \times \{nrecs = 3\}$$

$$e_{nDCG} : \theta_{agg} \times A_{agg}^{Test} \times \{nrecs = 3\} > e_{nDCG} : \theta_{os,p} \times U_{os,p}^{Test} \times \{nrecs = 3\}$$

Nevertheless, some users of the benchmark might focus on accuracy aspects, neglecting performance. In such cases, the corresponding functions are simply not applied during the evaluation process. Even more importantly, some of the goals may be conflicting and need to be prioritized by the given criteria. Consequently, a well-defined balance in the selection of suitable metrics to evaluate and compare, in other words, to benchmark a recommender SUT is of importance. This leads to a flexible, customizable, and comprehensible benchmark.

With the example in mind, the evaluation process can be further generalized to support the evaluation of various data aggregation functions and their resulting data sets. Therefore, initially, an aggregation function agg, as explained in Section 4.5, has to be applied to the available data $Data$. At this point, it must be noted that in contrast to the classical evaluation process, the prediction target of the recommender system has to be defined in advance. Basically, in the classical evaluation process, one signal type is used for training and testing the recommender system. For instance, training a recommender system based on purchase data aims to predict whether to recommend a product. Considering multiple signal types, this must not hold. This is based on the fact that the aggregated data sets incorporate various channels and their user signals. Consequently, the goal of a recommender system might range from predicting social media engagement to newsletter interactions or product purchases.

Then, first, a data splitting approach sp from a set SP of splitting approaches has to be selected and applied to the aggregated data sets A_{agg} to produce the corresponding train and test data sets A_{agg}^{Train} and A_{agg}^{Test}. Since the evaluation process to find an appropriate aggregation function may involve adapting the aggregation function, the same splitting approach sp has to be applied to achieve a fair comparison, as illustrated in the following:

$$sp : A_{agg} \mapsto (A_{agg}^{Train}, A_{agg}^{Test}).$$

Based on a given data set A_{agg}, various recommender system approaches rs of a set RS of recommender system approaches can be applied to the aggregated train data sets A_{agg}^{Train}. Additionally, a corresponding set of parameters P_{rs} for the selected recommender system approach rs has to be defined and applied, which leads to:

$$rs_{agg} : A_{agg}^{Train} \times P_{rs} \mapsto \theta_{agg,rs,P_{rs}}.$$

The result is a model $\theta_{agg,rs,P_{rs}}$ for the combination of the aggregated data set, recommender system approach and its parameters. As explained in the example, a performance metric m of a set M_p, for instance, CPU runtime or memory consumption, can be used to measure the model generation for a specific recommender system approach rs by:

$$e_m : rs \times A_{agg}^{Train} \times P_{rs} \mapsto \mathbb{R}_+.$$

Furthermore, a model $\theta_{agg_i,rs,P_{rs}}$ is evaluated by a generic evaluation function e considering a metric m of a set $M = M_a \cup M_p$ of metrics to measure its accuracy and performance based on a test data set A_{agg}^{Test} and a set P_{rs} of parameters of how to generate recommendations, as illustrated in the following:

$$e_m : \theta_{agg,rs,P_{rs}} \times A_{agg}^{Test} \times P_{rs} \mapsto \mathbb{R}_+.$$

Finally, the results of different recommender systems with varying parameter configurations and train data sets based on various aggregation functions have to be compared to find the most suitable one for the given application domain considering the available channels and signals.

Additionally, the evaluation process may involve comparing the results to an existing or already implemented recommender system rs_e, which is capable of dealing with the representation of the aggregated data sets. Considering such a recommender system as a black box, its recommendations could be used as a baseline to check whether a new approach makes sense regarding accuracy and performance.

4.6.5 Benchmark Components

System under Test

According to Section 3.6.1, a benchmark has to define the components comprising the SUT clearly and appropriately. Considering the introduced scenarios of the benchmark, the first component represents a storage system for the signals based on the data model in Section 4.2. The second component of the SUT is the considered recommender system approach. Consequently, the third component of the recommender system is the model of the recommender system. Another component is responsible for deploying the model and generate recommendations based on user queries.

Benchmark Driver

As illustrated in Figure 4.5, the benchmark driver orchestrates the SUT and the data generation based on a provided configuration of the benchmark user. Considering the SUT, the benchmark driver coordinates the steps in the benchmarking process. This includes to execute the loading of the data (Section 4.6.2) for the first workload and prepare it for the model training (Section 4.6.3) by applying the data aggregation and data splitting. Last, the benchmark driver takes care of requesting recommendations on the generated model.

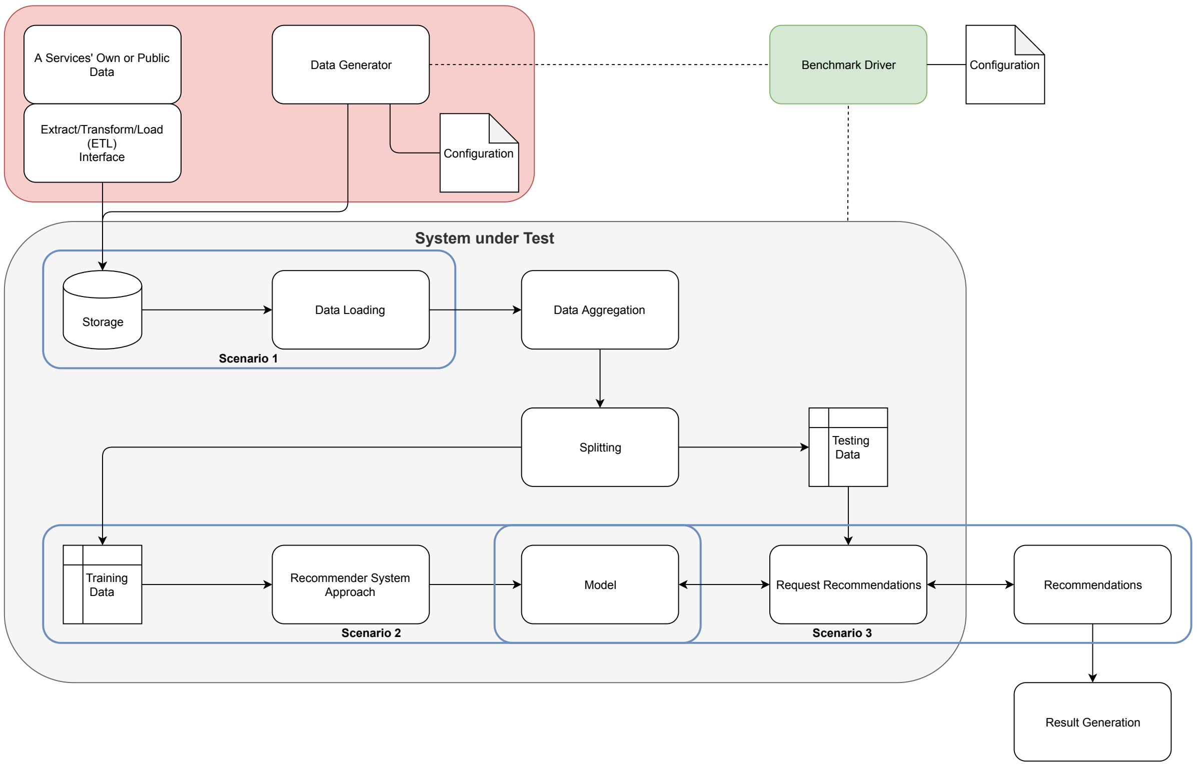


Figure 4.5: An overview of the benchmark concept. The benchmark driver (green rectangle) orchestrates all components and process steps during the benchmark execution based on a specific configuration. Top left (red rectangle) are the data components as discussed in Section 4.3. The SUT (grey rectangle) comprises data storage as well as the data aggregation (Section 4.5) and splitting steps. Furthermore, the recommender system approach (Section 2.4.1) and its model are part of the SUT. In addition, the three blue rectangles highlight the components and steps of the three scenarios.

Configuration

In this context, the configuration contains specifications for the data generation, such as the channels and the corresponding signals. For each channel, a distribution function has to be defined to generate a realistic number of signals, as discussed in Section 4.3.2. Additionally, the number of users and the number of items has to be specified. Furthermore, the configuration includes the applied data aggregation approach and its required parameters, such as the weightings for the channels (Section 4.5). The utilized splitting method is part of the configuration, too. Also, the recommender system approach and its required parameters are defined in the configuration. Besides, the configuration includes parameters to specify which metrics are determined during the execution. Finally, the number of recommendation requests for the users and their frequency is configured. To enable non-technical users to specify the benchmark, its configuration has to be provided in a readable format such as XML, JSON, or

YAML.

Results

Since the benchmark considers different aspects and consequently delivers multiple metrics, the representation of the results is important. In this sense, besides representing the results by numbers, it is also valuable to provide a visual representation, for instance, in the form of a radar chart. Furthermore, to support an analysis of the results, they have to be provided in a standardized form. Besides the results of the pure metrics, it also beneficial to provide the generated recommendations for the users. This enables the benchmark user to compare the recommender system approaches on a user basis, for instance, by checking how the generated recommendations differ. In this context, possible representations might be a relational database or a tabular format, such as an Excel sheet or CSV file.

5 Implementation of the Recommender System Benchmark

In this chapter, the benchmark concept introduced in Chapter 4 is explained and implemented. First, an overview of the modular implementation of the benchmark concept and how the modules relate to each other is given.

Then, the modules, namely, *data loading*, *data preprocessing*, *data aggregation*, *data splitting*, *algorithms*, *evaluation*, *configuration* and *visualization* are explained. This primarily involves elaborating on the implemented splitting and aggregation methods as well as the applied algorithms and metrics.

5.1 Overview

The benchmark implementation comprises a *data loading*, *data preprocessing*, *data aggregation*, *data splitting*, *algorithms*, *evaluation*, *configuration* and *visualization* module. To make the benchmark adaptable and extensible by a broad target audience, its implementation is based on *Python*. *Python* is easy to learn and, thereby, a widely-used programming language. This popularity especially holds for the field of data analysis and ML, to which recommender systems are addressed. Another advantage of *Python* is that it enables the benchmark to run on nearly every operating system, for instance, *Microsoft Windows*, *Linux*, or *MacOSX*. This fulfills the requirement for a benchmark to be portable.

Each module is responsible for one specific task. The *data loading* module provides functions to load the data from a source and transform it into the appropriate format. The *data preprocessing* module contains essential functions to manipulate the data. In the *data aggregation* module, functions to combine the different user signals are provided. In the *data splitting* module, functions to create train and test data sets are implemented. The *algorithms* module represents a wrapper to run already implemented recommender system algorithms. An evaluator in the *evaluation* module measures and updates the different metrics during the benchmark execution. Besides, a *visualization* module is available to save the benchmark results as figures. Finally, the *configuration* module represents a convenient way to load the benchmark and algorithm configurations. It is planned to publish the benchmark as an open-source project on GitHub as soon as it leaves its prototypical state. An overview of the described modules is depicted in Figure 5.1.

5.2 Modules

In the following section, the modules of the benchmark concept are described and explained.

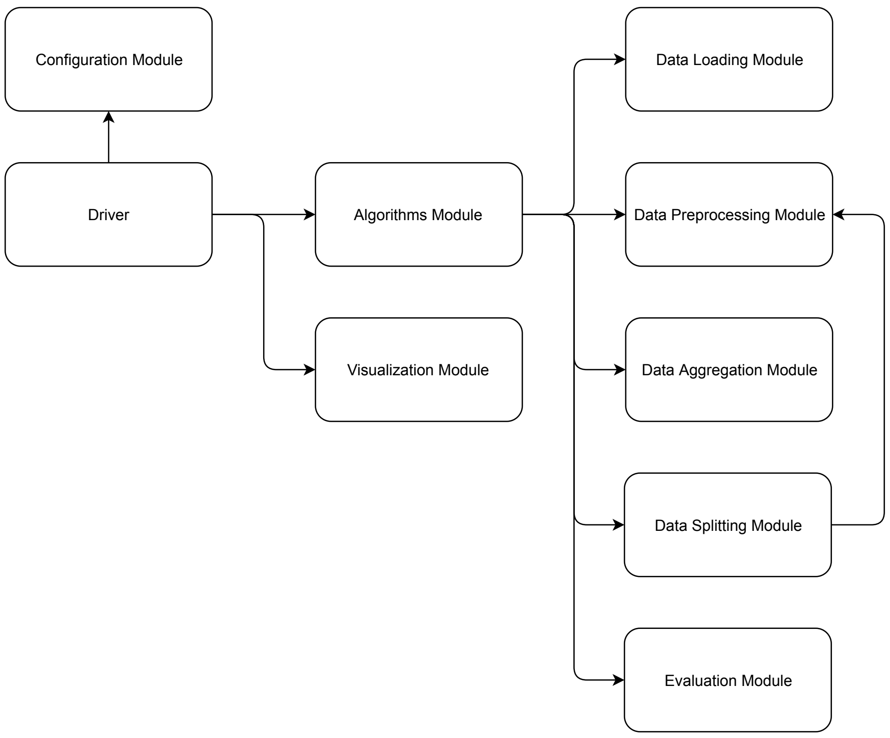


Figure 5.1: An overview of the modules and how they relate to each other. The arrows indicate the method call flow.

5.2.1 Data Loading

The data *loading* module is responsible for loading the required data from a specified data source. Therefore, a connection to the relevant database has to be established by this module. Due to the vast amount of different databases and interfaces, it might be required to customize the module's implementation to load the data. However, the effort for such an implementation is manageable since most databases provide *Python* libraries to connect to them. In the current implementation the loading is supported for the *MySQL* database and *CSV* files. The resulting format is a *Pandas* dataframe structure, which can be considered a table representation of the data.

5.2.2 Data Preprocessing

This module's functions provide easy manipulation and filtering of the data build on the *Python Pandas* library and its dataframe structure. First, this includes the transformation of the different data structures into appropriate data types. For instance, convert user and item information into numerical types and dates represented by strings into a corresponding date type. Furthermore, recommender system algorithms often require users and items to be stored sequentially. Therefore, the module implements functionality to support this task. Besides, functions to filter the data for a specific time interval or individual ratings are part of the module. Finally, it is possible to group users and items in the data set.

5.2.3 Data Aggregation

The data *aggregation* module provides two functions to aggregate the data, which are based on the idea elaborated in Section 4.5. The first function weights the different channels based on a given configuration. It is possible to specify an individual weight for a channel or even for a specific signal type (Section 4.5.3). The second function considers the timely occurrence of the user signals, keeping the latest for the further process (Section 4.5.4). To do so, the user signals are first sorted by their date, and then the duplicates are removed from the date, whereas the last one is kept. Here, a duplicate constitutes equal user-item signals. In doing so, the latest user-item signal remains in the data.

5.2.4 Data Splitting

The splitting methods implemented in this module are not part of most recommender system evaluations and are also not implemented in the libraries introduced in Section 2.7. The module contains two splitting methods, namely, *mask-based* and *time-based*. Their implementations are based on the explanations in Section 2.6.1 but are outlined in more detail here.

During the analysis of the online retailer data (Section 6.1.3), it became apparent that the same customer-product interactions appear multiple times. This is because some customers send a product back to the retailer, which resulted in negative values.

Another reason for that is the incorporation of additional user signals from other channels. Here, a customer might purchase and like an item leading to two user signals. Therefore, after a data aggregation method has been applied, in the case of *mask-based* splitting, all user-item signals are summed up to represent only one value. If the resulting value is negative, this user-item signal is neglected in the following, as depicted on the left-hand side of Figure 5.2.

In the case of *time-based* splitting the situation is slightly different, as illustrated on the right-hand side of Figure 5.2. Since the user signal dates are considered, the initial data set is first separated by a specified date. Then, for each data set, equal user-item signals are summed up and represented as one value. Finally, negative values are neglected.

A percentage value is used to split the data into a train and test set in both cases. For the *time-based* method, the percentage relates to the number of days within the configured time interval. Finally, the *splitting* module takes care of transforming the user and item names into a sequential numeric representation required by most recommender system algorithms applied in the following.

Besides, it is possible to define whether values in the train and test set should be considered as binary values. Binary values indicate relevance for an item and are, therefore, potentially more suitable for implicit user signals.

Applying the notation of the benchmark concept leads to the following set of splitting methods:

$$SP = \{\textit{mask-based}, \textit{time-based}\}$$

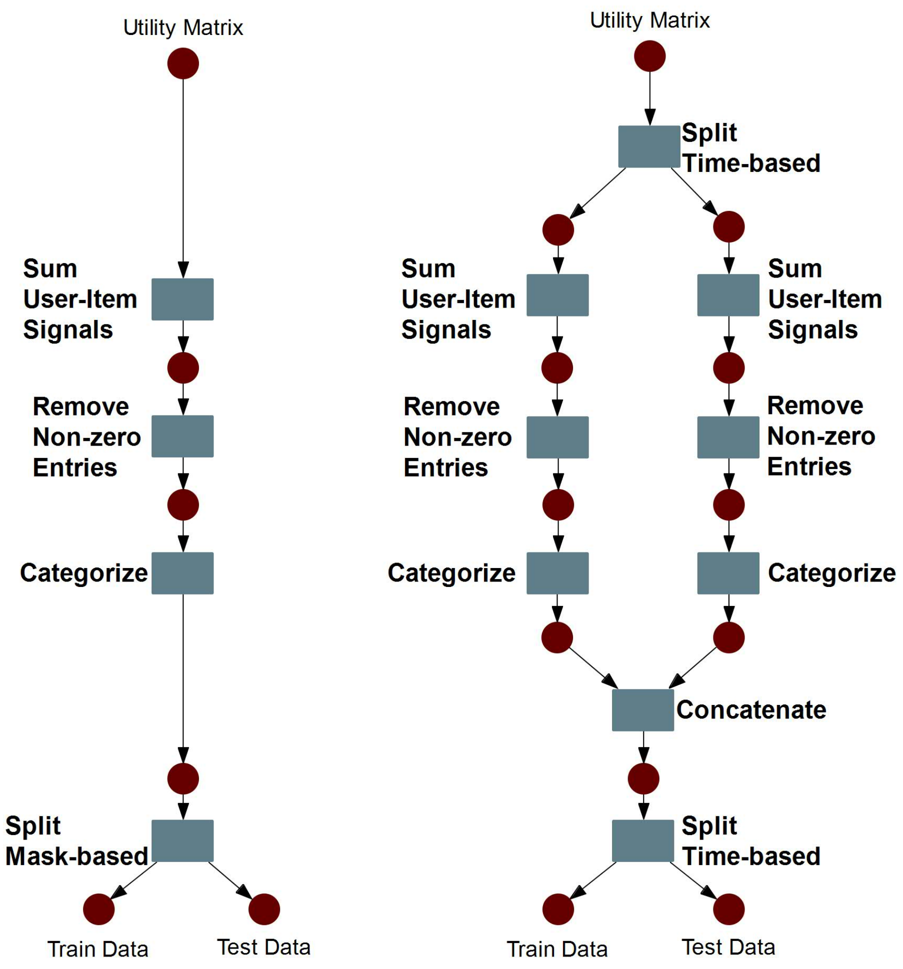


Figure 5.2: An overview of how the *mask-based* (left) and *time-based* (right) splitting methods process the initial utility matrix. The red object stores, which represent the matrix, are not named to keep the model clear. However, according to the activity name, the matrix changes. For instance, the activity *Remove Non-Zero Entries* generates a new matrix without zero entries.

5.2.5 Algorithms

The algorithms applied in this benchmark are *Alternating Least Squares (ALS)*, *Bayesian Personalized Ranking (BPR)*, *Item-based CF (II)*, *Singular Value Decomposition (SVD)* and *Co-Clustering (CoC)*. For the application of ALS and BPR, their *Python* implementations from the *Implicit* library are used. BPR is included additionally, since it is classified as an approach, which is specifically suited for implicit user signals, as discussed in [RFGST09]. Besides, the *LensKit* implementation of the II is part of the benchmark. Furthermore, the *Surprise* library provides the implementations of CoC and SVD, which are also used. Additionally, two baseline algorithms are applied, referred to as *Random (RAN)* and *Popularity (POP)*. In the former case, products are randomly selected for each customer from the product catalog and are afterward compared with the products purchased by the customers. In the latter case, the popularity of an item is based on the number of users who show a preference for it. In this approach, only one user preference contributes to the overall popularity of an item, although the user has shown multiple preferences by viewing, clicking, purchasing, or liking the item. Accordingly, the set of considered algorithms is defined as:

$$RS = \{ALS, BPR, II, SVD, CoC, RAN, POP\}$$

Furthermore, the benchmark focuses on the influence of the different aggregation methods considering the channels and signals; therefore, the default parameters of the considered algorithms are used in the benchmark. In Table 5.1 an overview of the parameters is given. An exception is the *center* setting of II, which is disabled in the default configuration for binary training, since a *centering* with 1-entries does not make sense. The algorithms are also defined to return a list of the ten most appropriate products for each customer. In other words, the top-N recommendations with $N = 10$ are considered, which reflects typical implementations of recommender systems.

Another important fact is that the algorithms, by default, are allowed to recommend items a user already showed an interest in in the past. This is based on the data analysis of an online retailer in Section 6.1.3, where about 8.5% of transactions have been repurchases of customers for the same product. This scenario might be unique for online retailing or in e-commerce but might not hold for other application domains. Consequently, this option can be defined in the configuration file of the benchmark. Nevertheless, a user of the benchmark can configure the algorithm parameters as he or she likes by changing the corresponding configuration files.

Table 5.1: An overview of the applied algorithms, the used abbreviations, their default parameters, the number of recommendations, and the utilized libraries. Here, f specifies the factor size, e the number epochs, l_r the learning rate, u_c the number of user clusters and i_c the number of item clusters.

Algorithm	Parameter Set	Top-N	Library
Alternating Least Squares (ALS)	$\{f = 100, e = 20\}$	10	Implicit
Bayesian Personalized Ranking (BPR)	$\{f = 100, e = 20, l\_r = 0.01\}$	10	Implicit
Item-based CF (II)	$\{k = 10,$ *center*$\}$	10	LensKit
Singular Value Decomposition (SVD)	$\{f = 100, e = 20, l\_r = 0.005,$ biased=True$\}$	10	Surprise
Co-Clustering (CoC)	$\{u\_c = 3, i\_c = 3, e = 20\}$	10	Surprise
Popularity (POP)	$\{\}$	10	LensKit
Random (RAN)	$\{\}$	10	LensKit

5.2.6 Evaluation

The following metrics are part of the *evaluation* module to benchmark the recommender system from a technical, business, and customer perspective. The technical metrics are *Response Time (T_{Res})* and *runtime* in milliseconds. In this regard, the *runtimes* of loading the required data (T_{Load}), training a model (T_{Train}), and testing a model (T_{Test}) are measured individually. The T_{Res} is measured by considering the time it takes for a model to generate the recommendations for a specific user. In this case, these are method calls of the utilized recommender system libraries. In the end, the average T_{Res} per user is determined by the evaluator. The customer perspective is covered by the metrics *nDCG* and *MRR* to evaluate the accuracy of the provided recommendations. From the business perspective, the *Catlog Coverage (CatCov)* is considered since it enables a company to check whether certain items are not relevant at all and mostly neglected by most evaluation libraries and frameworks. Furthermore, the evaluator discriminates between the absolute and relative *CatCov*. Denoted as $CatCov_a$ and $CatCov_r$, respectively. The $CatCov_a$ is the ratio of all items in the product catalog to the ones in the user recommendations, whereas $CatCov_r$ is the ratio of those in the test data to the ones in the user recommendations. To determine the metrics, the evaluator works on the generated recommendation list and the real user preferences given by the test data. Consequently, the accuracy, non-accuracy, and performance metrics discussed in Section 2.5 are defined as follows:

$$M_a = \{\textit{nDCG}, \textit{MRR}\}$$

$$M_{na} = \{CatCov_a, CatCov_r\}$$

$$M_p = \{T_{Res}, T_{Load}, T_{Train}, T_{Test}\}$$

Besides, the evaluator collects basic information about the considered data, such as the number of users and the number of items. Beyond that, for each algorithm, the number of unique items and the number of users receiving no recommendations are presented.

5.2.7 Configuration

The *configuration* module loads a configuration file for the execution of the benchmark. The chosen file format is *JSON* since it is commonly used as well as easy to read and maintain. Furthermore, it enables even non-technical users to run the benchmark since its execution is based on this configuration. In doing so, the benchmark satisfies the requirement to be easy to use. The configuration file contains parameters for the applied algorithms as well as a configuration for the benchmark. For the *load* module, it specifies the time interval of the data to consider. For the *aggregation* module, it specifies the method to use and how to weight the channels. For the *splitting* module, it contains the split method and the corresponding distribution of train and test data. In addition, it is possible to define if the train and test data should be binarized. The algorithm parameters are, for instance, the number of neighbors in case of *II*, the factor size in case of *ALS* and *SVD*, or the cluster sizes for *CoC*. It is also possible to specify whether it is allowed to recommend items a user already showed a preference for.

5.2.8 Visualization

The *visualization* module provides an easy way to analyze the benchmark results by generating figures for each considered metric depending on the applied algorithm. It generates bar charts where the x-axis denotes the algorithm, grouped by the split method, e.g., *mask-based* or *time-based*. The y-axis represents the corresponding metric, e.g., T_{Res}, T_{Load} T_{Train}, T_{Test}, *nDCG*, *MRR*, or the *CatCov*. The implementation is based on the *Python* library *Seaborn*.

6 Application of the Recommender System Benchmark

In this chapter, the benchmark is applied to a real-world data set from an online retailer covering the e-commerce application domain. This application extends the initial evaluation presented in [CHTV19].

At the beginning, an analysis of the real-world data of the online retailer is given. This analysis includes an overview of the available channels and signal types, as well as customer behavior and product information.

Then, the benchmark is applied, and its results are discussed from a technical, business, and customer perspective. Here, the technical perspective considers the loading time and response time. From the business perspective, the catalog coverage is examined. MRR and nDCG cover the customer perspective.

6.1 Analysis of the Online Retailer Data

This section provides some background information about the collaboration between the University of Münster and the online retailer, the channels and signal types, as well as an analysis of the provided data. Then, the applied data processing steps are explained, followed by a description of the considered data for the benchmark execution.

6.1.1 Collaboration Context

In 2016, the *ERCIS* and *Arvato CRM Solutions* initiated the project *ERCIS Omni-Channel Lab powered by Arvato*[1]. The collaboration combined research and teaching knowledge from academia with practical experience by an industrial business process outsourcing company. The overall goal of the collaboration was to investigate how to combine customer information distributed over several isolated systems in order to improve Customer Relationship Management (CRM) and customer service in particular. To do so, the team comprised researchers from the areas *statistics*, *processes* and *data* covered by the chairs *Information Systems and Statistics*, *Information Systems and Information Management* as well as *Computer Science*. During the project, the research team had the opportunity to work with a business partner of Arvato. The business partner was an online retailer interested in improving its recommender system.

[1]`https://omni-channel.ercis.org/`

6.1.2 Channels and Signal Types

The online retailer uses various channels to reach its customers. First, the online retailer offers its goods by classical brick and mortar stores. Here, the customers are tracked by the unique MAC addresses of their smartphones, which are broadcasted to the WiFi routers in the stores. Besides knowing what products a customer bought, this additionally enables the online retailer to record the time a customer spent in the store. Furthermore, an online shop is available to provide timely unrestricted and convenient access to its products. At this point, besides collecting customer purchases, the application of cookies enables to track a customer's browsing and search behavior. In the following, these signals are considered to be part of the channel *weblog* since they are collected by the retailer servers. Moreover, the online retailer uses social media to keep its customers updated by posting product promotions and campaigns, and general information on its Facebook channel. In doing so, the online retailer increases its customer's engagement by enabling the customers to like, comment, and share the provided content. Accordingly, these signals are assigned to the channel *social media*. Also, the online retailer regularly sends newsletters to its customers via email. In this regard, the online retailer collects whether a newsletter was sent to a customer, opened by a customer, or clicked by a customer. Opening and clicking a newsletter are considered as signals, which indicate an interest in its content. These signals are associated with the channel *email*.

6.1.3 Data Analysis

The data sets are provided as *CSV* files containing the catalog of the online retailer, customer tracking information of the stores, and purchases made by the customers. In addition, the weblog, as well as comments and likes on Facebook, are available. Furthermore, the newsletter interactions are part of the data set. In Figure 6.1 the time intervals of the data are depicted.

Products
The catalog data contains approximately 30,000 different products in total. Considering the interval of the available purchase data from June 2014 to December 2017, the number of products is roughly 11,000. A product is described by a unique identifier and metadata such as a description, product collection, product model, and product color.

Stores
Furthermore, the data set provides information about customers visiting the online retailer's brick and mortar stores. The data is collected by using the MAC addresses of the customers' smartphones during their visit. At this point, the time entering and leaving a store is recorded. The available data covers a time interval from May 2016 to May 2017. In this time interval, 3,500 customers with smartphones visited the stores. The average stay of a customer was approximately 230 seconds. A reason for such a short average duration might be given by the fact that people simply just pass the store without the goal of entering it. From the 3,500 collected customers 2,500 have a duration time larger than 0. Considering these customers, the average stay increases to 473 seconds, roughly 8 minutes. Unfortunately, the data collected as WiFi-Logs in the stores could not be utilized in the further process since it does not intersect

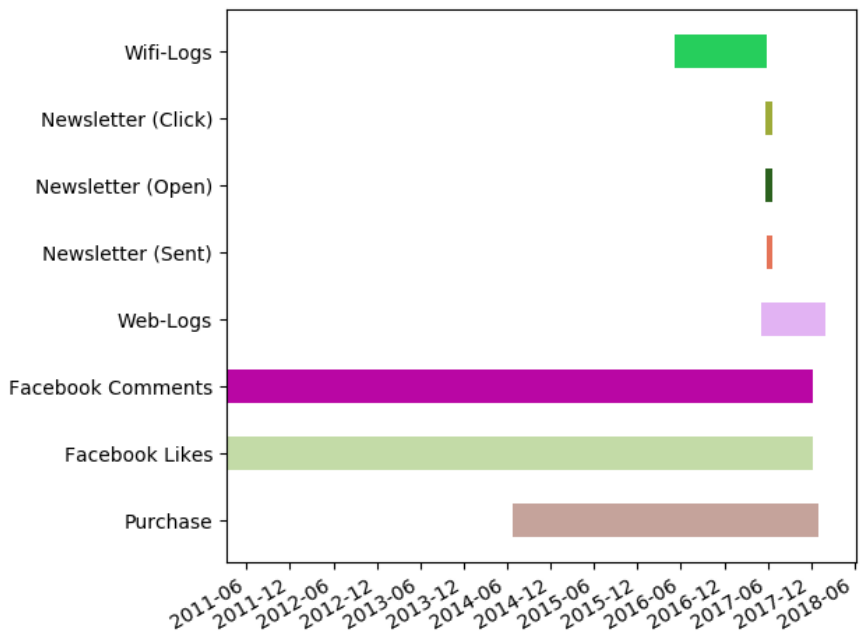


Figure 6.1: The available data sets and the time intervals they cover.

with the other data, as illustrated in Figure 6.1.

Purchases

The purchase data covers a time range from June 2014 until December 2017. In total, the data has $4{,}800{,}000$ entries with approximately $2{,}200{,}000$ orders, whereas an order contains one or multiple products and the purchased quantity. Considering identifiable customers, the data contains purchases of approximately $500{,}000$ customers for nearly $11{,}000$ different products. The maximum number of purchases is about $1{,}650$ and 1 is the minimum number. Roughly 20% of the $500{,}000$ customers purchased only one product of the catalog. On average, a customer bought about seven different products.

Weblog

The weblog data contains data collected by the server hosting the online shop of the online retailer. In this sense, it is a collection of browsing and searching behavior of the customers. The data covers a time range from May 2017 until January 2018 and contains $9{,}000{,}000$ entries in total. To gain useful insights from the data, it has to be processed, as will be explained in Section 6.1.4. The number of unique customers, determined by cookies, in the data set is $237{,}856$.

Social Media

The social media data contains likes and comments of customers on photos, videos, links, status, and notes on the company's Facebook page. Here, the total number of provided media content of the online retailer is $2{,}539$ in which $1{,}612$ are photos, 120 are videos, 405 are links, 398 are status, and 4 are notes.

Table 6.1: Two exemplary HTTP-requests illustrating the structure of the weblogs. The name of the online retailer as well as the product name have been replaced by *online-retailer* and *blanket*, respectively.

Type	HTTP-request
Website visit	https://www.online-retailer.com/blanket-model-number.html
Product search	https://www.online-retailer.com/?q=blanket

The likes data on social media covers a time range from March 2011 until December 2017. The total amount of customers liking one of the different media content types is approximately 25,000. The maximum number of likes made by a customer is 972, and the minimum number is 1. On average, a customer liked about five different media contents posted by the online retailer.

The comments data on social media covers a time range from March 2011 until December 2017. The total amount of customers commenting on one of the different media content types is approximately 1,500. The maximum number of comments made by a customer is 234, and the minimum number is 1. On average, a customer commented about three different media contents posted by the online retailer.

Email

Besides, the data set contains newsletter information classified into whether it was sent, opened, or clicked. At this point, the data covers a time range from May 2017 until June 2017. In this regard, about 254,000 customers received a newsletter, 68,000 opened it, and 15,000 clicked on its content. In total, 775,000 newsletters were send, 176,000 were opened, and 29,000 were clicked in that time. Furthermore, 10 different newsletters have been sent to the customers weekly in the considered time interval.

6.1.4 Data Preprocessing

Weblogs

To incorporate the weblogs as additional information into the recommender system, it must be preprocessed first. The weblogs collect all customer interactions in the online store. These, for instance, include a customer's browsing behavior as well as his or her product searches. To extract this information from the weblogs, regular expressions have been applied to the data to discriminate entries as either website visits or product searches and associate the results with products in the catalog. In Table 6.1 two exemplary entries illustrate the structure of the HTTP-requests. The customers in the weblogs are linked to the customers in the data by considering their devices' stored cookies.

Social Media

The available media content on the Facebook channel includes photos, videos, links, and status, which are liked or commented on by the customers. As discussed in Section 2.3 such signals would generally represent an explicit customer signal since the customer explicitly likes or expresses his or her opinion about the provided media content. In this scenario, this does not hold since the provided media content includes various products. On Facebook, a company can tag specific products within a photo. For instance, a posted photo may illustrate a living room with a couch, a blanket, and a candle from the catalog. Consequently, a like of a customer cannot be associated with exactly one product. Nevertheless, Facebook provides an API to extract the tagged products within the photo. Utilizing this API, the product names are extracted and associated with the catalog by applying a string matching approach. Besides, videos and links are associated with products considering their descriptions. A simple name matching achieves the association of customers using Facebook to the customers in the database. The described method can extend the purchase data by implicit feedback of customers using Facebook.

Newsletter

Moreover, the interaction of customers with newsletters is utilized as an additional data source. Unfortunately, it is unclear which product of the newsletter has been clicked by a customer, but only that the newsletter was clicked at all. In this case, a newsletter's header information is extracted and associated with possible products in the catalog by string matching. This leads to an indication of interest in the products, but with less confidence.

Product Matching

As explained for weblogs, social media, and newsletters, a preprocessing step is required to benefit from these additional sources since no direct association with a product is given. To generate this association, the extracted strings are compared to products in the catalog. For this reason, the string matching approach *token set ratio* has been applied. Compared to similarity measures such as the Levenshtein distance, the token set ratio works for strings of different lengths and word orders. Furthermore, it also supports partial matches. In the process, a string is first separated into words and then split into an intersection and two remainders, as illustrated for the words X and Y in the Equations 6.1, 6.2 and 6.3. As indicated by **sort**, each set is ordered alphabetically to handle different word orders. The pairwise similarity is defined as $\frac{2 \cdot M}{T}$, where M is the number of matches and T the number of characters.

$$t_0 = sort(X \cap Y) \tag{6.1}$$

$$t_1 = t_0 + sort(X \setminus t_0) \tag{6.2}$$

$$t_2 = t_0 + sort(Y \setminus t_0) \tag{6.3}$$

An example of the sets t_0, t_1 and t_2 for X="blue blanket" and Y="grey blanket" is given in Table 6.2. Here, the pairwise comparisons lead to the following scores: $sim(t_0, t_1) = 0.74$, $sim(t_0, t_2) = 0.74$ and $sim(t_1, t_2) = 0.75$. Considering the former score, it is determined as $\frac{2 \cdot 9}{12+12}$ where 12 is the length of t_1 and t_2. The 9 matching characters are based on the match of the term "blanket" in t_1 and t_2 having 8 common characters as well as the partial match of "e" in "blue" and "grey".

Table 6.2: The following example illustrates the application of the token set ratio on the words X="blue blanket" and Y="grey blanket".

Tokens
t_0=sort({"blue", "blanket"} ∩ {"grey", "blanket"})="blanket"
t_1="blanket" + sort({"blue", "blanket"} \ {"blanket"})="blanket blue"
t_2="blanket" + sort({"grey", "blanket"} \ {"blanket"})="blanket grey"

6.1.5 Considered Data

The time interval covering most of the channels' available data is from May 2017 to January 2018. Considering the mentioned time interval, the considered data contains the following information, as illustrated in Figure 6.2.

Purchases

In this case, the total number of purchases in the data set is 643,409, which are part of 316,790 orders. Considering the identifiable customers and purchased products, the data set contains 170,000 and 6,000, respectively.

After removing products with negative quantities from the data and grouping and summing by customer-product combinations, the number of interactions was 588,168. From the 170,000 customers nearly 56,000 purchased only one product.

Weblog

The time range of the weblogs covers the considered time interval, such that the previously provided information is still valid. The number of preference indicators could be increased by applying the string token ratio approach on the server log files.

After doing this, the number of website visits inferred by the weblogs was 2,410,111, and the number of searches 48,127. After grouping and summing by customer-product interactions, there were 325,329 website visits and 7,511 searches.

Social Media

For the time between May 2017 and January 2018, roughly 25,000 entries are available from the likes data. These entries cover likes of 232 different media contents from approximately 6,700 customers.

The number of entries in the comments data is 760. Here, the comments are placed on approximately 232 media content from roughly 402 customers.

Applying the set token ratio, the total number of inferred preferences by comments and likes was 5,834 and 390,828, respectively.

In the specified time interval, likes contributed 189,559, and comments contributed 45 additional preference indicators after grouping and summing by customer-product combinations.

Email

The number of sent, opened, and clicked newsletters remains the same as described in Section 6.1.3. Here, 544,075 preferences could be inferred from opened newsletters and 28,359

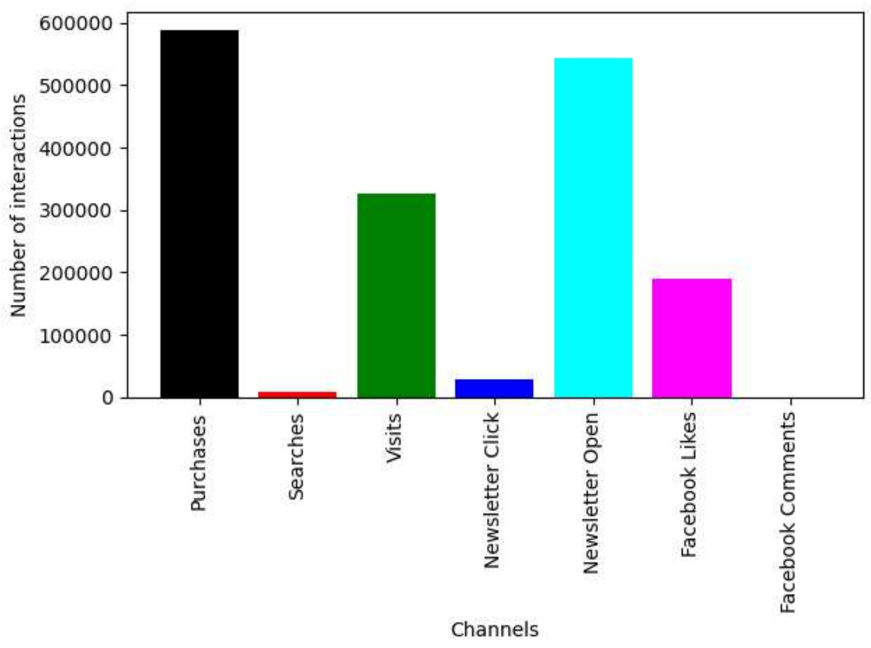


Figure 6.2: The inferred preference indicators among the provided channels.

from clicked newsletters.

6.2 Application of the Benchmark to the Online Retailer Data

In this section, the different workloads are executed and the results are presented. Before that, the general setup of the benchmark is explained, which includes the utilized notation and the specification of the testing environment.

6.2.1 General Setup

Notation

At this point, the available channels Ch_{or} and signal types S_{or} are summarized in Table 6.3. Here, the or denotes the online retailer. Additionally, the abbreviations of the channels and signals used in the further process are listed in Table 6.4. In this regard, $*, *$ denotes a shorthand notation for all channels and all signal types.

In addition, the algorithms are abbreviated as *Alternating Least Squares (ALS)*, *Bayesian Personalized Ranking (BPR)*, *Item-based CF (II)*, *Singular Value Decomposition (SVD)*, *Co-Clustering (CoC)*, *Popularity (POP)*, and *Random (RAN)*.

Table 6.3: The table summarizes the channels and signal types provided by the online retailer.

Entity	Value
Ch_{or}	{online shop, brick and mortar, weblog, social media, email}
S_{or}	{purchase, view, search, comment, like, visit, open newsletter, click newsletter}

Table 6.4: Overview of the channels and their corresponding signal types used by the online retailer. Furthermore, their abbreviations are listed. To represent all channels and signal types, the shorthand notation $*, *$ is used.

Channel	Signal type	Abbreviation
online shop	purchase	os,p
brick and mortar stores	visit	bm,v
	purchase	bm,p
weblog	view	wl,v
	search	wl,s
social media	comment	sm,c
	like	sm,l
email	open newsletter	em,o
	click newsletter	em,c
all	**all**	$*, *$

Furthermore, the labelling of the algorithms in the resulting figures has to be explained, which makes use of the following pattern:

$$\{alg\}\_\{os\}\_\{wl\}\_\{em\}\_\{sm\}\_\{agg\}\_\{b\}$$

for which

$$als\_40\_30\_20\_10\_w\_b$$

may serve as an example in the following sense: The first letters represent an abbreviation for the algorithm, for instance, *ALS*. Then, the used weighting of the considered channels and the applied aggregation method follow, for instance, 40 for online shop, 30 for the weblog, 20 for email, and 10 for social media. At this point, w stands for the weighting-based aggregation, whereas a s for the sequence-based. At the end, the b indicates whether the model training is based on a binary representation of the data.

Testing Environment

The testing environment specifies the hardware resources the SUT was deployed on. For the benchmark execution, a single system is utilized. The operating system installed on the system is *Windows 8.1 Enterprise Edition (*64*-bit)*. Besides, the system includes an *Intel(R) Core(TM) i7-6700 CPU* with 8 Cores and 16 GB main memory. Furthermore, an *Intel(R) HD Graphics* 530 *graphic card* is part of the system.

For extracting, transforming, and loading the *CSV* files *Pentaho* was used. The database access during the benchmark execution has been simulated through the usage of a locally stored *CSV* file containing all needed customer and product signals. Besides, *Python 3.8.5* is installed with the following libraries: *Pandas (1.1.0)*, *NumPy (1.19.1)*, *Cython (0.29.21)*, *Implicit (0.4.2)*, *LensKit (0.10.1)* and *Surprise (1.1.1)*. The *Pandas* library is used to load the *CSV* file.

Configuration

In the following benchmarks, some configurations are always applied in the same way and should be mentioned. The first configuration relates to the distribution of the training and testing data. Considering the *mask-based* splitting, the training data contains 80% of the initial data. For *time-based* splitting, the training data covers 80% of the days between the minimal and maximal date within the initial data. Furthermore, in all cases, the testing data is given in binary form. This representation means that it only contains zero or one values where one indicates a product's relevance for a customer. For the training data, different configurations are applied and will be explained for each benchmark separately. Besides, for all benchmarks, the time interval from 2017-05-05 to 2018-01-30 is considered. To recall, the applied metrics by the benchmark are summarized in Section 5.2.6 and explained in Section 2.5.

6.2.2 Binary Aggregation on Purchase Data

This approach investigates the channels *online shop* and *brick and mortar store* and the signal type *purchase*.

Applying this to Equation 4.7 leads to:

$$w(u, i, Data, \{os, bm\}, \{p\}, \text{2017-05-05}, \text{2018-01-30}) = \sum_{(u,i)\in Data, ch\in\{os,bm\}, s\in\{p\}} f_{ch,s}(u, i, c)$$

Here, the function $f_{ch,s}(u, i, c)$ returns the amount of purchases. In the following, all aggregated values $w(u, i)$ larger than 0 are set to 1.

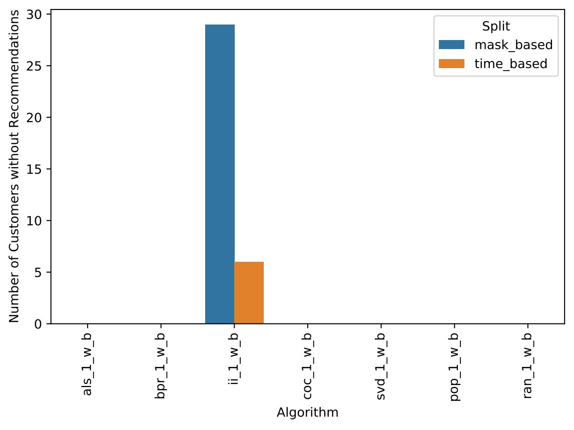
Result Analysis

In case of *time-based* splitting the number of customers in the test set are $16{,}828$ and $59{,}960$ in case of *mask-based* splitting. In both cases, the number of products is $5{,}787$.

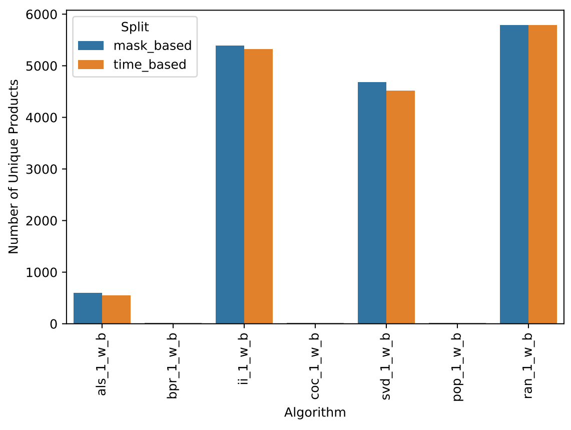
Considering the number of customers without recommendations in Figure 6.3a, only *II* stands out by leaving 29 and 6 customers without any recommendation. Nevertheless, this issue could be circumvented by extending the recommendation lists with random or popular products. The other algorithms are able to generate recommendations for all customers, as depicted in Figure 6.3a. Considering the number of different products among all customer recommendation lists in Figure 6.3b, the baseline algorithms show the expected results. The

RAN approach recommends each product of the catalog at least once. This leads to the best result since a high value is preferred. In contrast, the *POP* approach is limited to recommend the 10 most popular products. The second-highest number of different products is provided by *II*, followed by *SVD* for both splitting methods. In this setup, *CoC* and *BPR* perform equally poor, with only 10 different products.

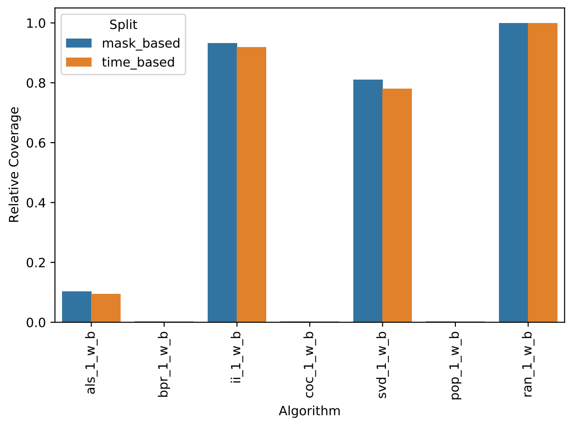
The catalog coverage underlines the gained findings, as illustrated in Figure 6.3c and Figure 6.3d, where the relative coverage is the ratio of the number of different products in the recommendation lists to the number of products in the data set. In contrast, the absolute coverage is the ratio of the different products in the recommendation lists to the number of all products in the retailer's catalog. It shows, that *POP*, *BPR* and *CoC* have the lowest catalog coverage. The best performance is achieved by *RAN*, followed by *II*, which performs well for *time-based* and *mask-based* splitting with a coverage of 93% and 91%, respectively. Then, *SVD* follows with 80% and 78% and *ALS* with approximately 10% in both cases.



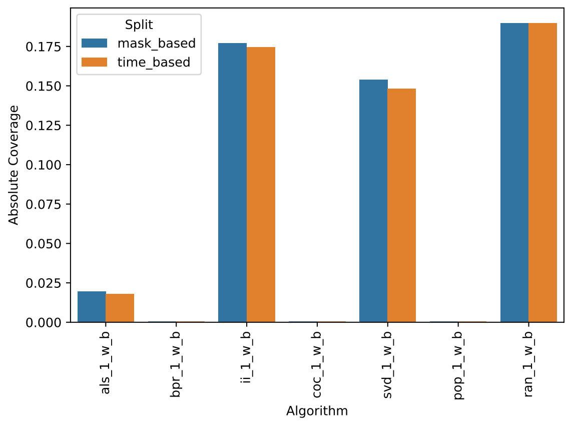
(a) Customers without recommendations.



(b) Unique products.



(c) Relative coverage.

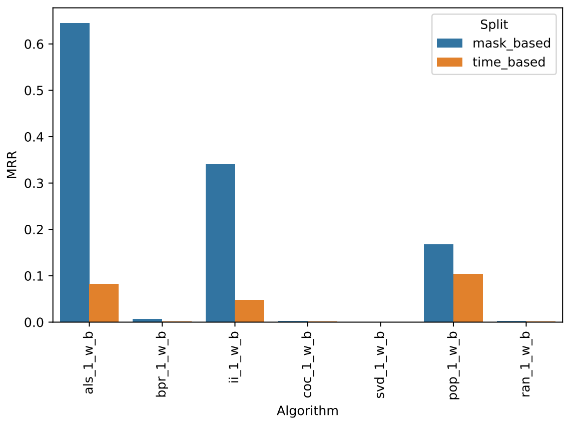


(d) Absolute coverage.

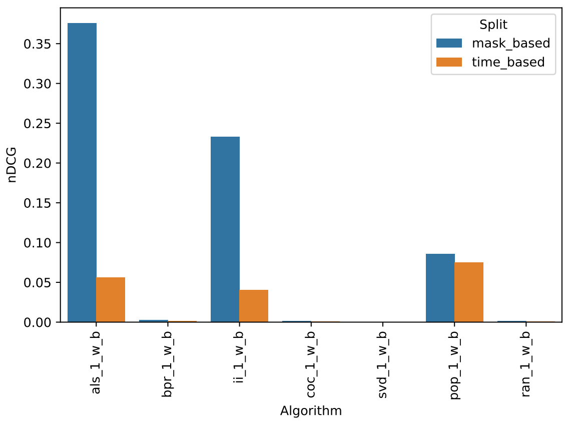
Figure 6.3: An overview of the customer and product results based on the purchase data.

Considering the accuracy metric MRR of the recommendation, *ALS*, *II* and *POP* show the best results for both splitting methods as indicated in Figure 6.4a. To recall its interpretation, the higher the MRR, the higher the first appearance of a relevant product in the recommendation list is observed.

The best performance for *mask-based* splitting is given by *ALS* with a value of 0.65. In case of *time-based* splitting, *POP* performs best, with a value of 0.10. The ranking quality of the complete recommendation list is measured by its nDCG. Applying this metric shows no difference in the order of the results, as illustrated in Figure 6.4b. Still *ALS* outperforms *II* and *POP* in case of *masked-based* splitting, whereas *POP* is slightly better in case of *time-based* splitting.



(a) MRR.



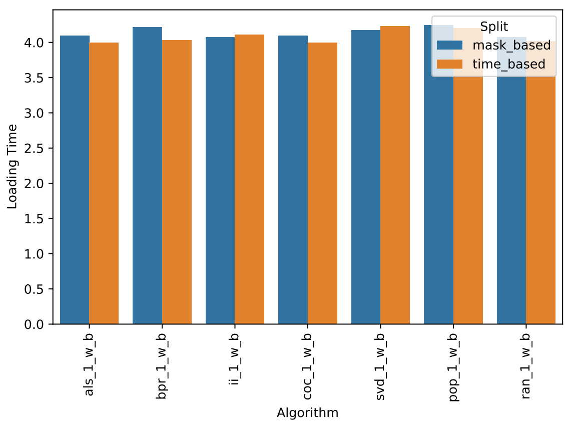
(b) nDCG.

Figure 6.4: An overview of the accuracy results based on the purchase data.

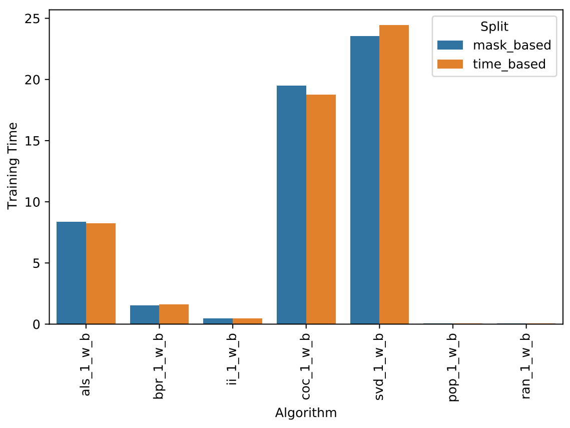
Finally, the runtimes of the algorithms are investigated. As expected, the time to load the data is similar since all algorithms apply the same query to retrieve it from the local *CSV* file, as shown in Figure 6.5a.

Considering the training time of Figure 6.5b, *POP* and *RAN* are the fastest with less than 0.01 seconds, for both splitting methods. Then, *II*, *BPR* and *ALS* are following. Considering the applied splitting methods, the training time is stable for each algorithm. This is actually not the case for the testing times in Figure 6.5c, since the number of tested customers differs between the two methods. In this case, the testing time for *mask-based* splitting is higher than for *time-based* splitting. The lowest testing time is given by *ALS* with 56 and 17 seconds, followed by *BPR* with 63 and 17 seconds. Then, *II* follows with 270 and 77 seconds.

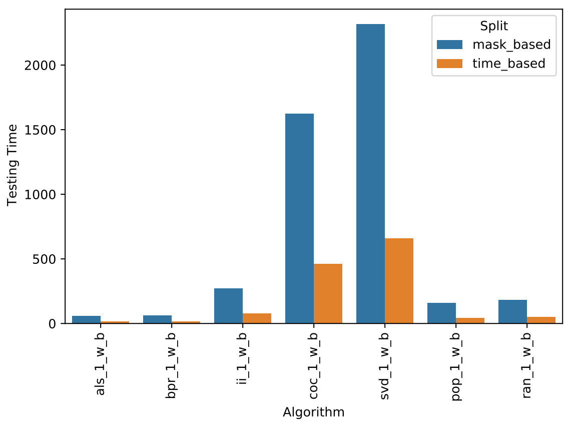
The implementations of *CoC* and *SVD* perform poorly in this regard with testing times higher than 1,600 seconds for *mask-based* splitting and 460 seconds for *time-based* splitting. The models' response times underline the observed testing times, as illustrated in Figure 6.5d. The lowest response time is given by the *ALS* and *BPR* with less than 0.2 seconds, followed by *POP* and *RAN*. Far behind are *CoC* and *SVD*.



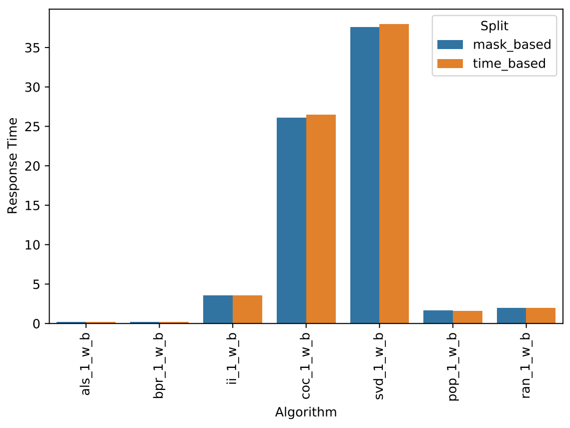
(a) Loading Time (sec).



(b) Training Time (sec).



(c) Testing Time (sec).



(d) Response Time (ms).

Figure 6.5: An overview of the timing results based on the purchase data.

Conclusion
Considering the technical perspective, *ALS* and *BPR* perform best. They are trained fast and have a low response time, but *ALS* outperforms *BPR* in the other metrics, such as product coverage and accuracy. From the business perspective *II* can recommend many products from the catalog. Taken accuracy into account *ALS* outperforms *II* clearly in case of *mask-based* splitting but only slightly for *time-based* splitting which rather represents a real-world scenario. In this sense, depending on the importance of the response time, both represent suitable solutions.

6.2.3 Binary Aggregation on Omni-Channel Data

This approach investigates all channels and all signal types.

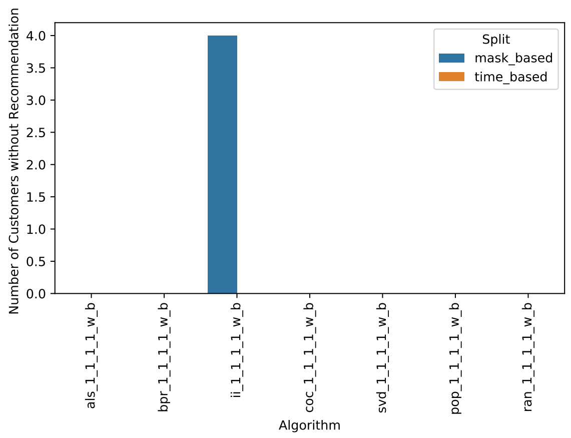
Applying this to Equation 4.7 leads to:

$$w(u, i, Data, *, *, \text{2017-05-05}, \text{2018-01-30}) = \sum_{(u,i) \in Data,*,*} f_{*,*}(u, i, c)$$

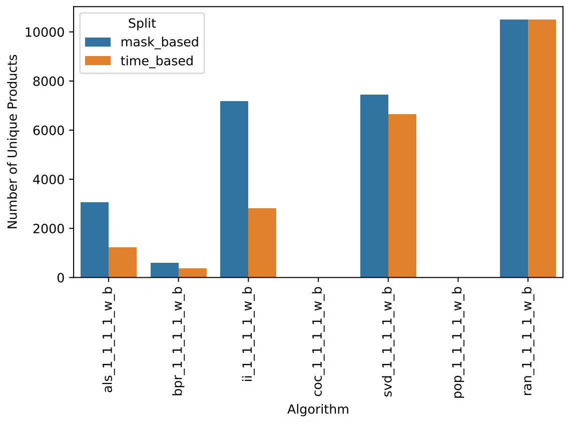
In this case, the function $f_{*,*}(u, i, c)$, depending on the channel ch and signal type s, returns the number of purchases, the number of views and searches, the number of likes and comments or the number of newsletter interactions. Then, for each customer-product combination, the returned values are summed up to an overall preference value. Afterward, for each non-zero value of $w(u, i)$, the final value is set to 1, motivating the name binary aggregation.

Result Analysis

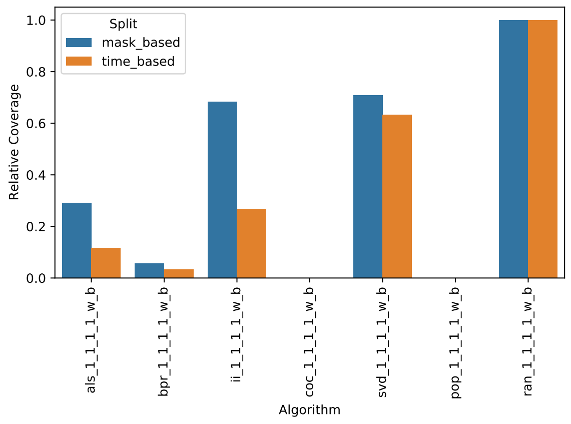
In case of *time-based* splitting the number of tested customers is $10{,}796$ and $61{,}703$ in case of *mask-based* splitting. In both cases, the number of products is $10{,}502$.



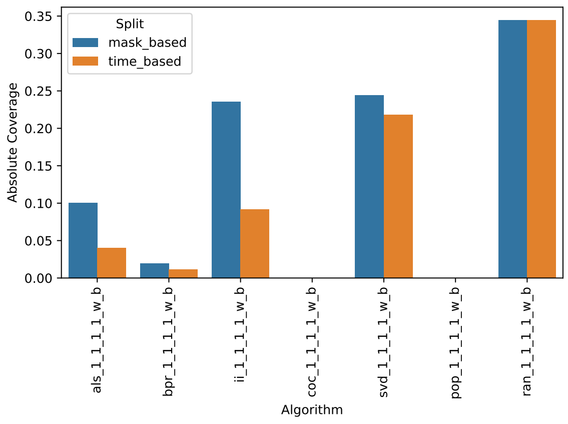
(a) Customers without recommendations.



(b) Unique products.



(c) Relative coverage.



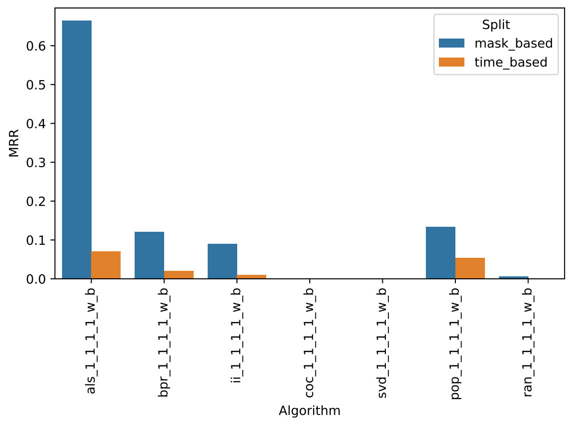
(d) Absolute coverage.

Figure 6.6: An overview of the customer and product results based on the complete data.

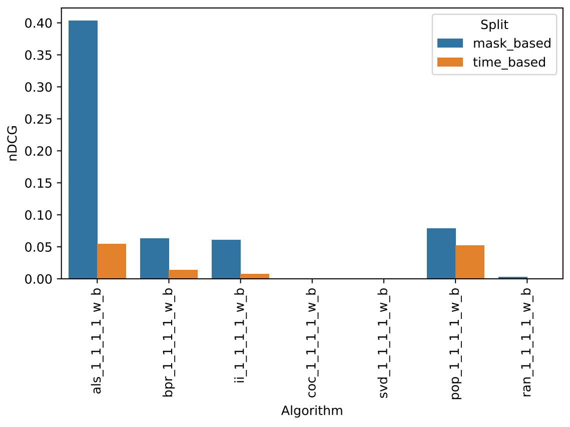
Comparing the number of customers without recommendations in Figure 6.6a, only *II* is not able to generate recommendations for 4 customers. The highest number of unique products is given by *RAN* with $10{,}502$, as shown in Figure 6.6b, followed by *SVD* and *II* with $7{,}434$ and $7{,}172$ for *mask-based* splitting, respectively. In case of *time-based* splitting, the difference of the two implementations is higher with $6{,}655$ for *SVD* and only $2{,}807$ for *II*. Then, *ALS* and

BPR are following. Here, both implementations provide more different products in case of *mask-based* splitting, with 3,052 compared to 1,218 and 597 compared to 361. The lowest performance is given by *CoC* with 10 products. Consequently, the number of unique products is reflected by the product coverage, as shown in Figure 6.6c. Regarding the products in the data set, *RAN* recommends each product at least once. For *mask-based* splitting, *II* and *SVD* show a relative coverage of approximately 70% and an absolute coverage of roughly 24% in Figure 6.6d.

Considering the accuracy of the implementations, *ALS* shows the best MRR and nDCG results based on *mask-based* splitting, as shown in Figure 6.7b. Then, far behind, *BPR*, *II* and *POP* follow. In case of *time-based* splitting, *POP* is able to compete with *ALS*, with a nDCG of 0.052 compared to 0.054. The results of *POP* and *ALS* are in line with the MRR results with 0.053 and 0.07, respectively. Comparing *BPR* and *II*, they show similar nDCG results, but *BPR* performs better considering its MRR value of 0.12 compared to 0.09, as shown in Figure 6.7a. To recall, this means that a relevant product appears high in the recommendation list.



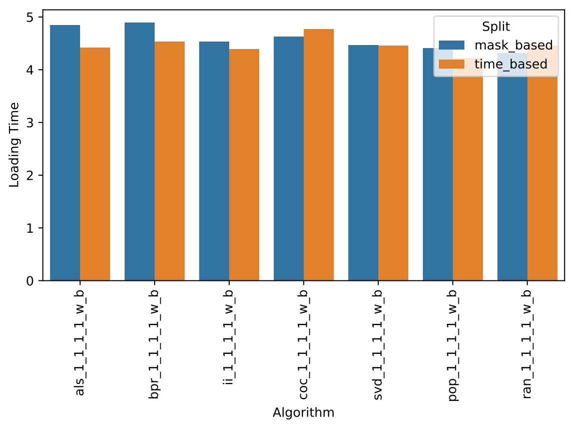
(a) MRR.



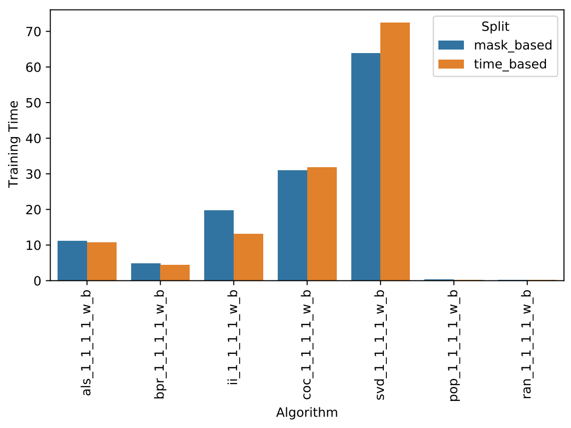
(b) nDCG.

Figure 6.7: An overview of the accuracy results based on the complete data.

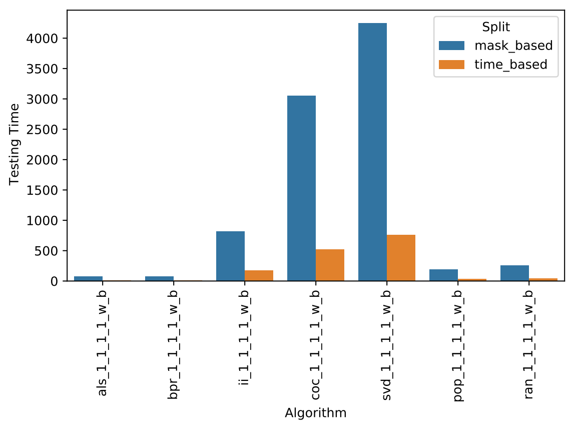
The loading times of the algorithms are quite similar, as illustrated in Figure 6.8a. The shortest training times were achieved by *POP* and *RAN*. Compared to the other algorithms, *SVD* has the longest training times for both splitting methods, as shown in Figure 6.8b. Generally, the best training time is achieved by *BPR*, followed by *ALS*, *II* and *CoC*. The measured testing times in Figure 6.8c show that the testing time of *SVD* is the longest with 4,250 seconds. The lowest testing times are given by *ALS* and *BPR* with roughly 10.8 and 4.5 seconds for *time-based* splitting, respectively. They even outperform the testing times of *RAN* and *POP*. The performance of *II* is acceptable compared to *ALS*, but *CoC* is nearly 20 seconds slower. The response times in Figure 6.8d show the superiority of *BPR* and *ALS* with response times between 0.23 and 0.34 milliseconds. Neglecting *RAN* and *POP*, the next best response time is given by *II* with 12 milliseconds.



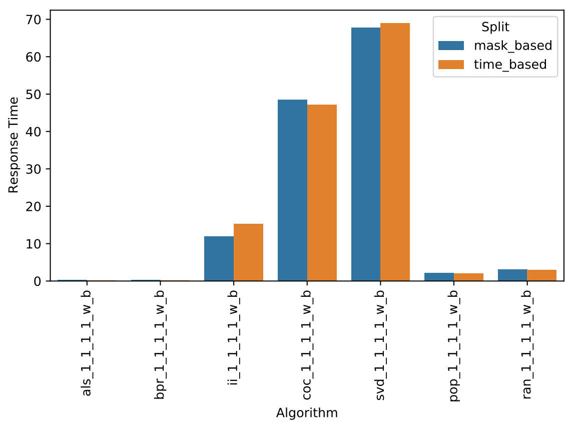
(a) Loading Time (sec).



(b) Training Time (sec).



(c) Testing Time (sec).



(d) Response Time (ms).

Figure 6.8: An overview of the timing results based on the complete data.

Conclusion

Considering training time, testing time and response time, *RAN* and *POP* perform best. However, the former lacks in accuracy and the latter only covers a small portion of the catalog. Besides, *BPR* has a short training time, testing time, and response time but only recommends a few different products. In this sense, the most suitable implementations are *ALS* and *II*. Considering the product coverage, *II* performs better, such that it is a suitable approach from a business perspective. However, the provided accuracy of the *ALS* is much higher, leading to satisfying results for the customers. Depending on whether accuracy or catalog coverage is preferred either *II* or *ALS* are reasonable solutions.

6.2.4 Weighting-based Aggregation on Omni-Channel Data

This approach investigates all channels and all signal types occurring in the time interval from 2017-05-05 to 2018-01-30. Besides, this approach is referred to as weighting-based aggregation since the customer signals are weighted depending on the channel.

Applying this to Equation 4.7 leads to:

$$w(u, i, Data, *, *, \text{2017-05-05}, \text{2018-01-30}) = \sum_{(u,i) \in Data,*,*} \alpha_{ch,*} \cdot f_{*,*}(u, i, c)$$

In this case, the function $f_{*,*}(u, i, c)$, depending on the channel ch and signal type s returns, for instance, the amount of purchases, the number of views or searches, or a like of the customer-product combination. Each customer-product is summed up to an overall preference value. At this point, three different weighting configurations have been applied:

$$(\alpha_{*,p} = 25, \alpha_{wl,*} = 25, \alpha_{em,*} = 25, \alpha_{sm,*} = 25) \tag{6.4}$$

$$(\alpha_{*,p} = 40, \alpha_{wl,*} = 30, \alpha_{em,*} = 10, \alpha_{sm,*} = 20) \tag{6.5}$$

$$(\alpha_{*,p} = 40, \alpha_{wl,*} = 20, \alpha_{em,*} = 10, \alpha_{sm,*} = 30) \tag{6.6}$$

At this point, it must be noted that the weightings are absolute values. Furthermore, the values are not optimized for any of the considered implementations. The values instead represent an intuitive selection of weighting the signals on the channels. For instance, in each configuration, the individual absolute weightings on each channel sum up to 100. In Weighting 6.4 all channels are treated as equally important such that each of the four channels is weighted by 25. Next, in Weighting 6.5, the purchases are considered as the most important customer signal followed by views and searches in the online shop. Then, social media and newsletter are following. Finally, in Weighting 6.6, only the importance of social media and views and searches changes.

Result Analysis

In case of *time-based* splitting the number of tested customers is 10,796 and 61,703 in case of *mask-based* splitting. In both cases, the number of products is 10,502.

As already observed by the other approaches, *II* is the only algorithm that is not able to generate recommendations for all customers. However, the number varies based on the applied weightings. In case of Weighting 6.4, 234 customers remain without any recommendation, whereas applying Weighting 6.5 or Weighting 6.6 leads to 65 and 63 customers without recommendations for *masked-based* splitting. Considering the number of unique products, *ALS* provides approximately 2,500 different products with Weighting 6.5 and Weighting 6.6. This covers about 28% of the products in the data set and 10% of the complete catalog. In contrast, Weighting 6.4 leads to about 2,300 different products covering 26% of the data set and 9% of the catalog. In case of *II*, Weighting 6.4 and Weighting 6.5 generate the highest number of different products with at least 1,300, whereas the number is 1,236 for Weighting 6.6. Here, the relative coverage is 12% as shown in Figure 6.9 and the absolute coverage is 4%. The order of the highest to lowest number of different products is independent of the splitting method. Considering *SVD* and *BPR*, no influence of the weightings is observable. In contrast, *CoC* performs quite poor but is able to double the number from 10 to 20 with Weighting 6.4.

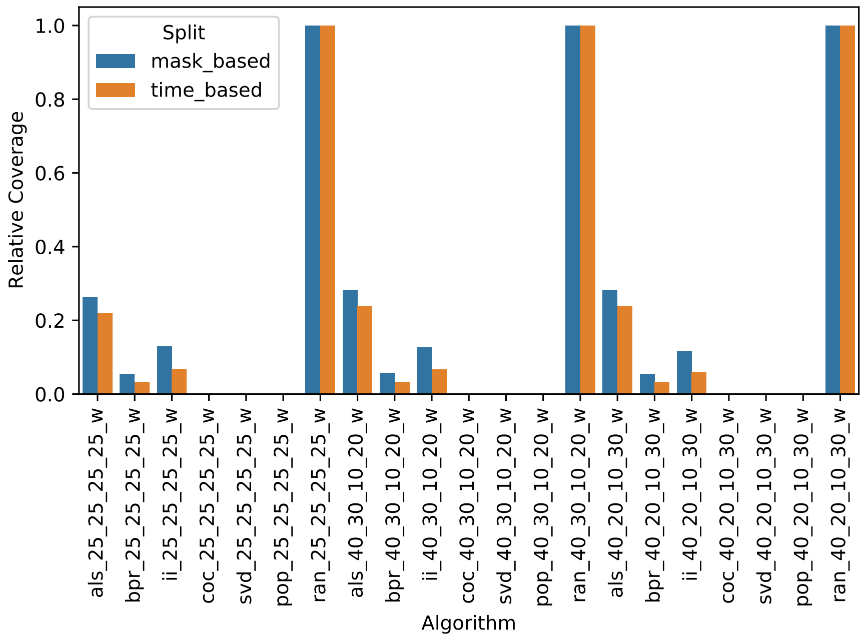


Figure 6.9: An overview of the relative coverage results based on the weighted data.

Considering the accuracy of the recommendations in Figure 6.10 and Figure 6.11, the nDCG value improves slightly for *ALS* from 0.70 to 0.72 when Weighting 6.5 or 6.6 are applied in case of *mask-based* splitting.

For *time-based* splitting, the situation is the other way around, but the differences are quite small. However, the MRR aligns with the results of nDCG, showing the same order.

Considering *II*, *SVD* and *CoC*, they show very low nDCG and MRR values and are even beaten by *RAN*. Besides *ALS* and *POP*, only *BPR* achieves reasonable nDCG results with a value of 0.055 for Weighting 6.4, 0.051 for Weighting 6.6 and 0.047 for Weighting 6.5 in case of *mask-based* splitting. The values of MRR show the same order.

The weightings themselves show no influence on the training times, testing times, and response times of the different implementations. Regarding the training time, *POP* and *RAN* still perform best, followed by *BPR*, *ALS* and *II*.

The lowest response times are performed by *BPR* and *ALS*, followed by *RAN* and *POP*, whereas *CoC* and *SVD* show the highest response times, as illustrated in Figure 6.12. Compared to *CoC* and *SVD*, the implementation of *II* also achieves acceptable results.

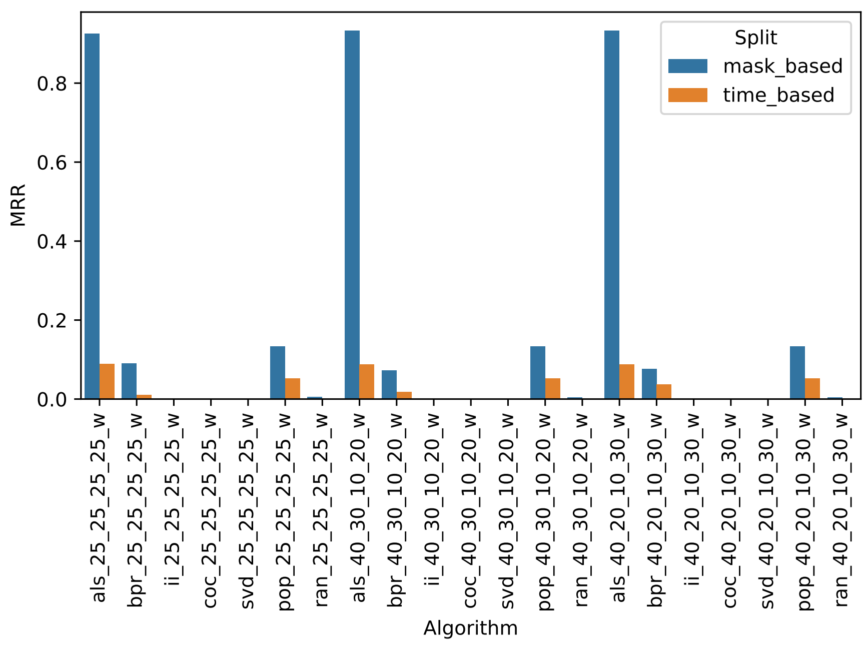


Figure 6.10: An overview of the MRR results based on the weighted data.

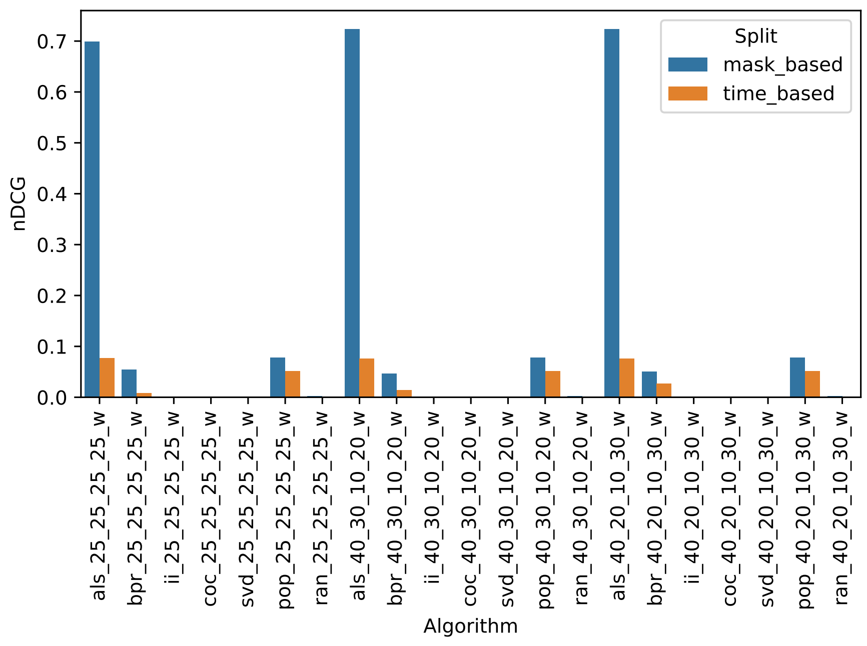


Figure 6.11: An overview of the nDCG results based on the weighted data.

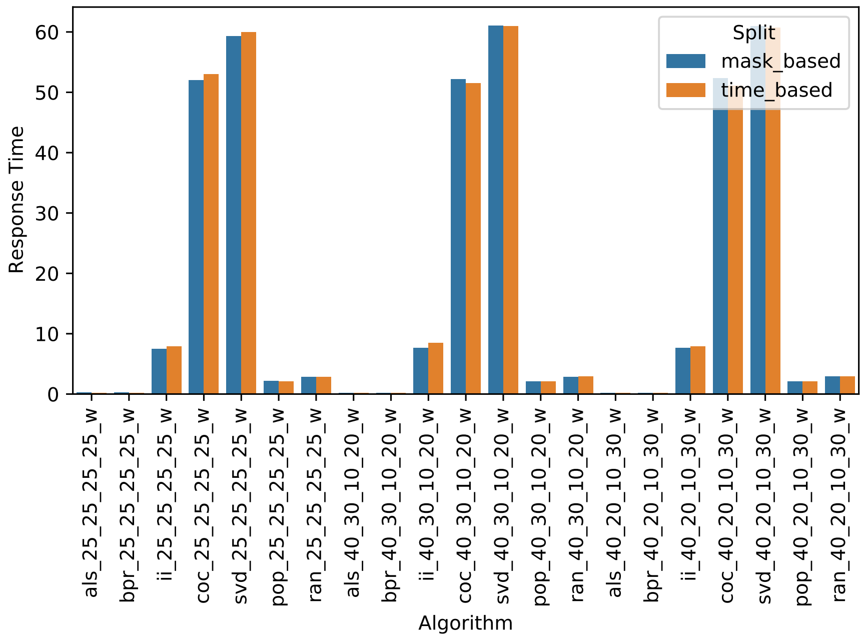


Figure 6.12: An overview of the response time (ms) results based on the weighted data.

Conclusion

The three selected weightings show only a small influence on the results. However, *ALS* achieves a better accuracy and higher product coverage when the Weightings 6.5 and 6.6 are applied. This might be a result of weighting purchases with a factor of 40 instead of 25. Furthermore, an improvement for *II* is given by the catalog coverage. To sum up, in this scenario *ALS* clearly outperforms the other implementations. The reason for this is that, in contrast to the other implementations, *ALS* is optimized for implicit feedback provided by the customers.

6.2.5 Sequence-based Aggregation on Omni-Channel Data

In the sequence-based aggregation, for all channels and signal types, only a customer's last signal is kept regardless of its signal type.

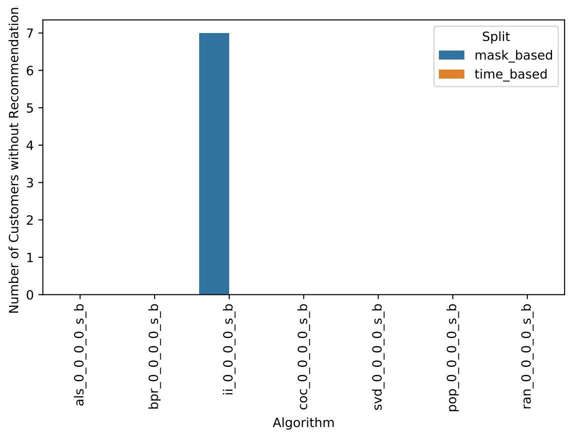
Applying this to Equation 4.6 leads to:

$$w(u, i, Data, *, *, \text{2017-05-05}, \text{2018-01-30}) = \underset{\forall (u,i) \in Data,*,*}{\textit{latest}} f_{*,*}(u, i, c)$$

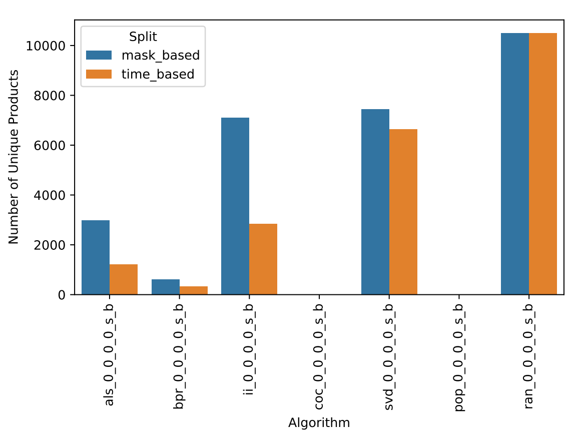
In this case, only the last indication of interest for a product is considered relevant since it might represent a customer's most recent preference for it. Here, the latest occurrence of customer signals is considered for the time between 2017-05-05 and 2018-01-30.

Result Analysis

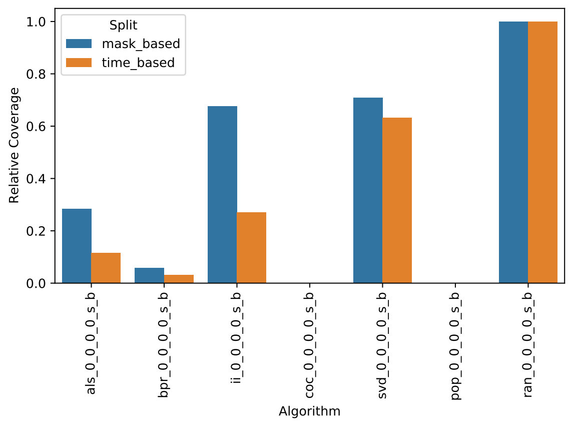
For *mask-based* splitting the number of customers is $61{,}876$ and for *time-based* $10{,}590$. The number of products for the two splitting methods is the same with a value of $10{,}095$. Again, only *II* is not able to recommender products for 7 customers for *time-based* splitting, as illustrated in Figure 6.13a. The highest number of unique products is achieved by *RAN*, whereas *CoC* and *POP* provided the lowest value with only 10. In case of *mask-based splitting*, the next best value is achieved by *SVD* with 7,454 followed by *II* with 7,106 products. Then, with a bigger gap, *ALS* and *BPR* are following, as shown in Figure 6.13b. Considering the *time-based* splitting, the gap between *SVD* and the other algorithms is larger. For instance, *SVD* provides 6,643 products and *II* as third best only 2,840. These results are further underlined by the relative product coverage in Figure 6.13c. Here, *SVD* covers at least 67% of the products in the test data catalog, representing nearly 25% of the complete catalog, as shown in Figure 6.13d. Only the implementation of *II* is able to keep up with it for *mask-based* splitting. In contrast, the highest relative coverage that *ALS* achieves is 28%, which corresponds to 10% of the complete catalog.



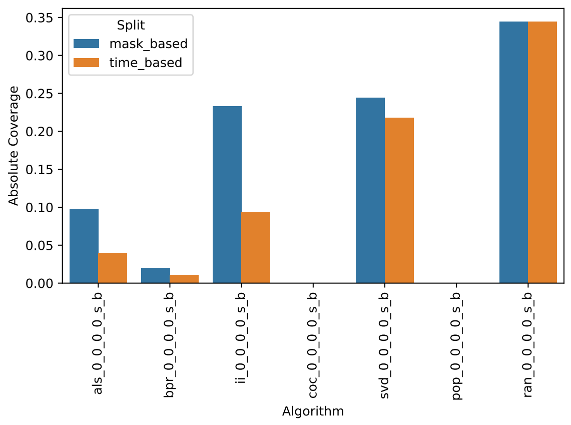
(a) Customers without recommendations.



(b) Unique products.



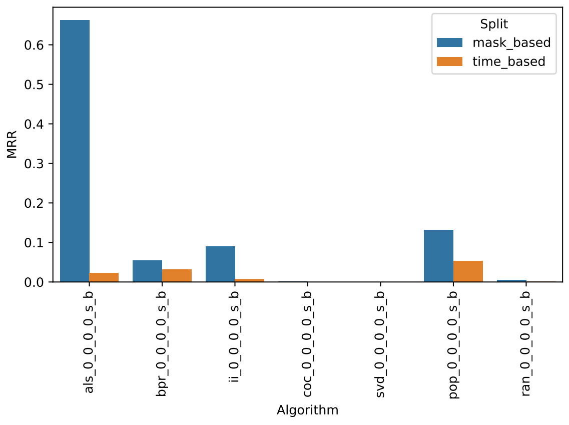
(c) Relative coverage.



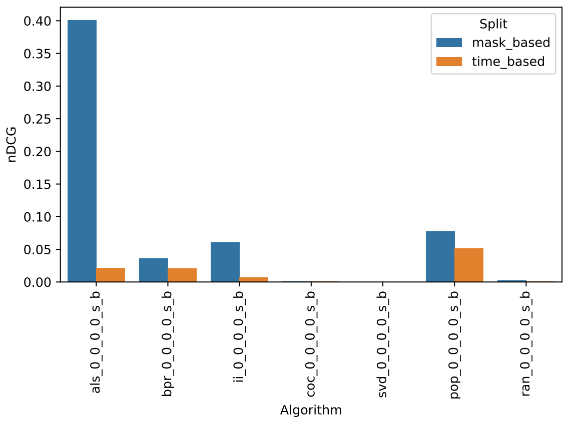
(d) Absolute coverage.

Figure 6.13: An overview of the customer and product results based on the sequence-based data.

With regard to the accuracy metrics in Figure 6.14a and Figure 6.14b *ALS* clearly outperforms the other algorithms when *mask-based* splitting is applied. In this case, its nDCG value is 0.4 and the second best, achieved by *POP*, is 0.077, followed by *II* with 0.06. However, in the case of *time-based* splitting, the achieved results are much closer. Now, *POP* performs best with a value of 0.05, followed by *ALS*, *BPR* and *II*. The lowest performances are achieved by *RAN*, *CoC* and *SVD*. The MRR metric in Figure 6.14a shows similar results and thereby supports the results of nDCG. This metric further underlines the high accuracy of *ALS* in case of *mask-based* splitting.



(a) MRR.



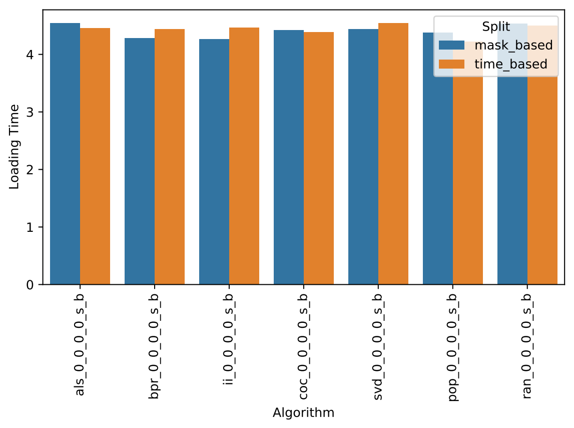
(b) nDCG.

Figure 6.14: An overview of the accuracy results based on the sequence-based data.

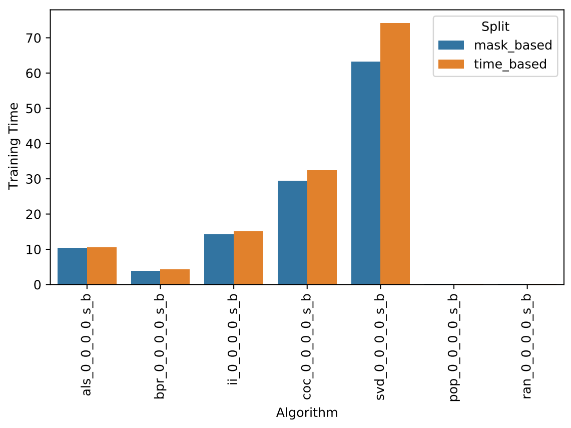
Since the algorithms work on the same data, the loading times do not differ considerably among them, as shown in Figure 6.15a. Considering the training time in Figure 6.15b, *BPR* shows the shortest runtime for both splitting methods with about 4 seconds, when *RAN* and *POP* are ignored. Then, *ALS* and *II* follow with approximately 10.5 and 14.5 seconds, respectively. The longest training time is given by *SVD*, which requires nearly double the time for training as *CoC*, roughly 60 against 30 seconds for *mask-based* splitting.

Taking a look at the testing time in Figure 6.15c, *SVD* shows the poorest performance of the algorithms, followed by *CoC*. In particular, for the *mask-based* splitting, since more customers are tested.

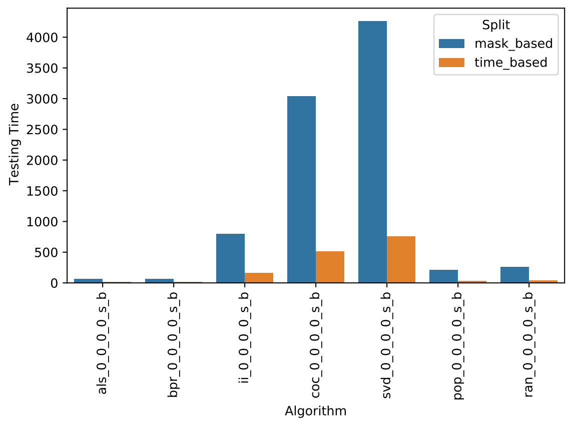
After *SVD* and *CoC*, *II* follows with a lower time of 800 seconds for *mask-based* and 163 seconds for *time-based* splitting. The shortest testing times are achieved by *ALS* and *BPR* with about 60 seconds for *mask-based* and 10 for *time-based* splitting. These numbers are in line with the response times of the models to generate recommendations shown in Figure 6.15d. Here, *ALS* and *BPR* show their strength of fast recommendation generation. The next best times are achieved by *POP* and *RAN*. The highest response times are given by *CoC* and *SVD*, whereas *II* is placed in between.



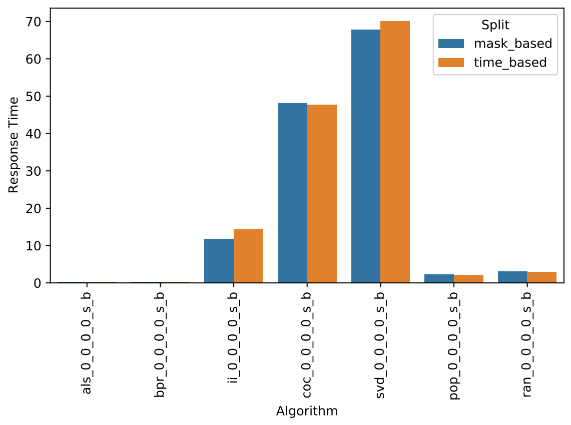
(a) Loading Time (sec).



(b) Training Time (sec).



(c) Testing Time (sec).



(d) Response Time (ms).

Figure 6.15: An overview of the timing results based on the sequence-based data.

Conclusion

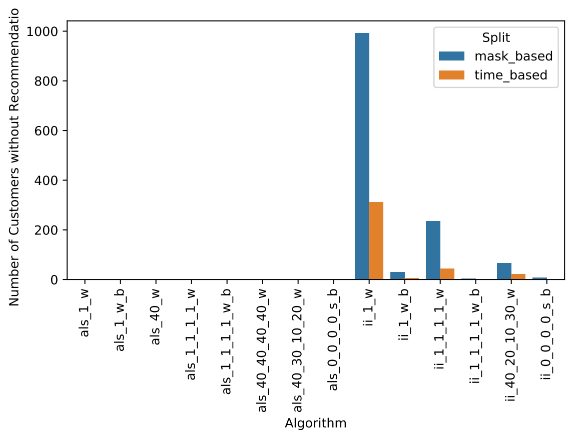
Considering *SVD*, it shows the best product coverage values but a low accuracy, which makes it a less preferable option. On the other hand, *POP* also shows a low coverage, but a good accuracy for *time-based* splitting. This underlines the impression that the recommendation of currently trending and popular products provide a simple but still useful heuristic. Also, the runtime is acceptable, making it an option to generate fast recommendations for, e.g., trending products. The best overall performance from a customer and technical perspective is provided by *ALS* with *mask-based* splitting, but with the downside of lower coverage. In this sense, *II* could represent an alternative if coverage is more important.

6.2.6 Overall Analysis

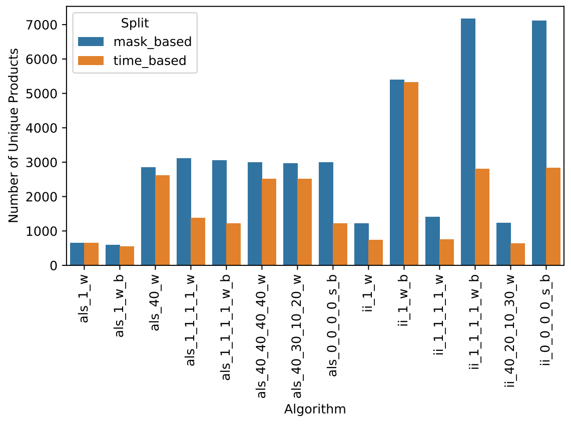
In the previous section, different algorithms have been benchmarked against each other under consideration of a specific aggregation method. In this section, the best results of the different aggregation methods are compared. Therefore, since *ALS* and *II* showed the best results, these two are examined in the following. For a fair comparison, three other evaluation results are part of this analysis. This includes the application of *ALS* and *II* directly on the purchase data denoted as *als_1_w* and *ii_1_w* as well as the application on the complete channel data denoted as *als_1_1_1_1_w* and *ii_1_1_1_1_w*. Furthermore, *als_40_w* has been added. This is because a weighting of 40 is a proposed value in [HKV08, p. 4]. The other *als_40_30_10_20_w* is the best *ALS* solution considering nDCG in the weighting-based aggregation, whereas *ii_40_20_10_30_w* is the best for *II*.

Results Analysis

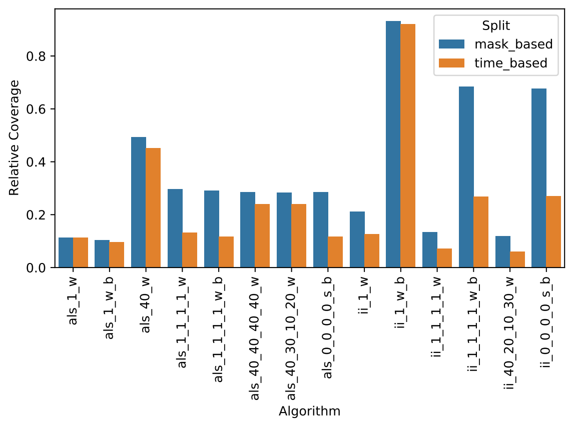
Considering the number of customers without recommendation, *II* shows a benefit by using the other channels, since the number is lower in case of *ii_1_1_1_1_w_b* and *ii_0_0_0_0_s_b*. The other algorithms provide recommendations in all cases, as illustrated in Figure 6.16a.



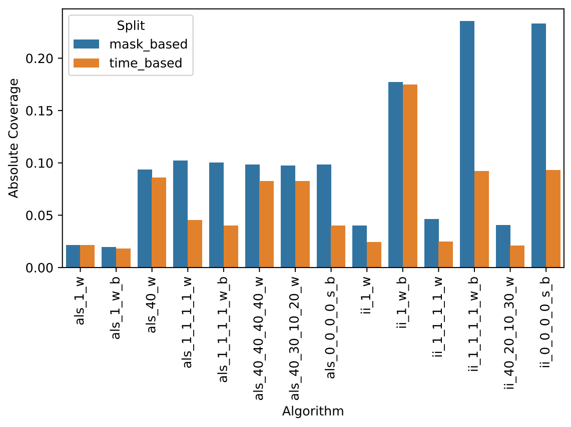
(a) Customers without recommendations.



(b) Unique products.



(c) Relative coverage.



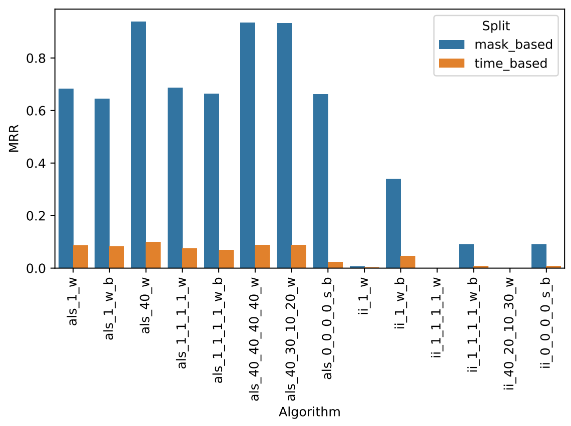
(d) Absolute coverage.

Figure 6.16: An overview of the overall customer and product results.

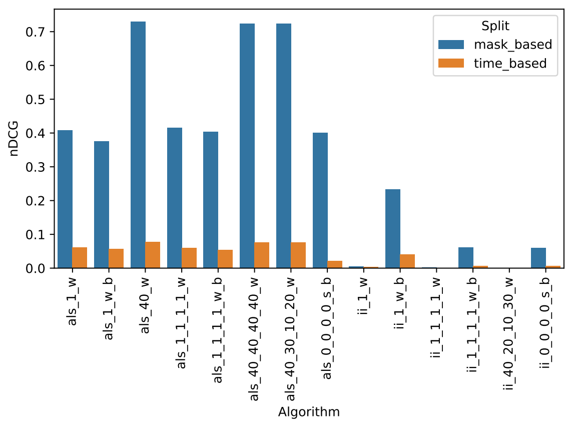
Comparing the number of unique products provided by the algorithms, *II* outperforms the others in case of *mask-based* splitting, when the other channels are used, regardless of the applied aggregation method as shown by *ii_1_1_1_1_w_b* and *ii_0_0_0_0_s_b* in Figure 6.16b.

Nevertheless, the second-best result is achieved by *ii_1_w_b* on the binary purchase data without using the channels. Considering the best results of *ALS*, in case of *mask-based* splitting the results are slightly better when additional channels are added, 2,848 compared to 3,113. As shown in Figure 6.16c, the best relative coverage result is achieved by *ii_1_w_b* with a value of 93%, followed by two other configurations with additional channel information, whereas the absolute coverage in Figure 6.16d is lower compared to *ii_1_1_1_1_w_b* and *ii_0_0_0_0_s_b*. In the case of *ALS*, the best relative coverage is given with a weighting of 40 without additional channel information. Taking a look at the *sequence-based* and *weighting-based* aggregation for *ALS* they show no positive effect to the relative product coverage, but a slight one for the absolute coverage, as shown in Figure 6.16d.

Next, the accuracy of the aggregation methods is examined. As shown in Figure 6.17b, *ALS* already shows a very high accuracy without including additional channel information into the data set. The quality is high whenever the purchase data is weighted with a value of 40, *als_40_w*, *als_40_40_40_40_w* and *als_40_30_10_20_w*. Considering the *time-based* splitting, the differences are lower, but still, no positive effect in the recommendation quality is visible. The same holds for *II*, where the initial data set provides the best results and the values only considered by their relevance in binary form as shown by *ii_1_w_b*, *ii_1_1_1_1_w_b* and *ii_0_0_0_0_s_b*. Generally, *II* cannot compete with the nDCG values achieved by *ALS*, especially, in the case of *mask-based* splitting. The nDCG results correspond to the MRR results shown in Figure 6.17a.



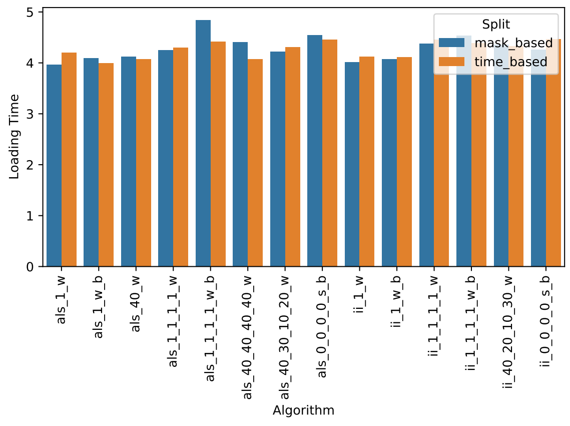
(a) MRR.



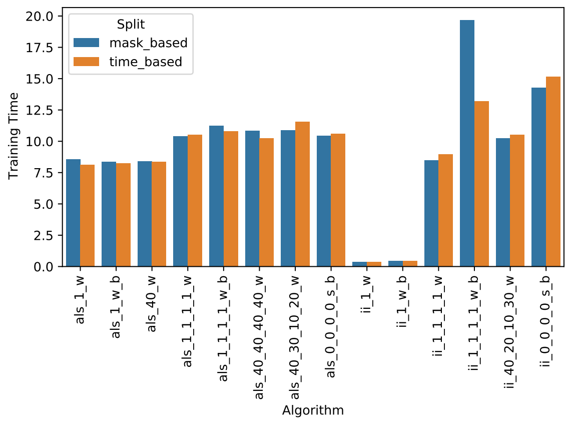
(b) nDCG.

Figure 6.17: An overview of the accuracy results.

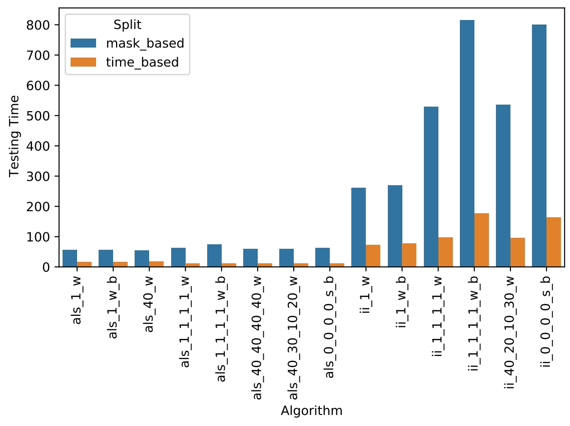
Since the data loading is based on a local CSV file the corresponding loading times in Figure 6.18a are quite similar among the different aggregations. Considering the training time, the results of *II* vary, as shown in Figure 6.18b. Here, the training time is influenced by the higher number of products when additional channels are part of the data set, which increases from 5,787 to 10,502. In the case of *ALS* this influence is also visible in the results. For instance, by comparing *als_40_w* and *als_1_1_1_1_w_b*. When the same aggregation is used, no influence is observable. The testing time of *II* is longer compared to *ALS*, as illustrated in Figure 6.18c. Again, testing more customers leads to longer testing times. In Figure 6.18d, the response time results show the high performance of *ALS*. Nevertheless, again, the influence of the additional channels is visible. Even *ALS* shows slightly higher response times when all channels are used. In this case, the response times are larger than 0.22 milliseconds, whereas in the other case, they are less than 0.18 milliseconds.



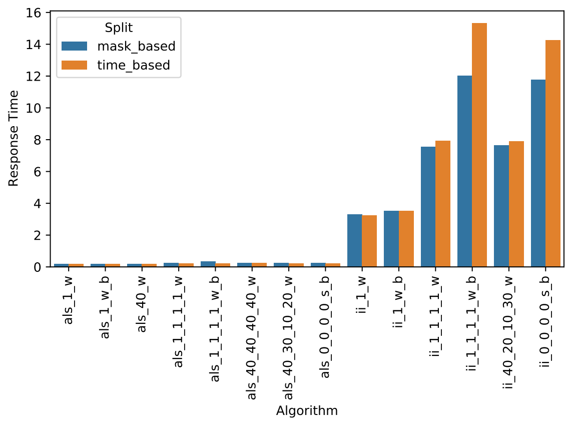
(a) Loading Time (sec).



(b) Training Time (sec).



(c) Testing Time (sec).



(d) Response Time (ms).

Figure 6.18: An overview of the overall timing results.

For the investigated weightings, no positive influence in the resulting recommendation accuracy was measurable so far. In the following, for *ALS* and *II* various weighting combinations are tested with the *masked-based* splitting method. At this point, the weighting-based aggregation, discussed in Section 6.2.4, is applied. In the case of *II*, 75 combinations are tried based on individual channel weightings. Here, the weightings are defined as:

$$\alpha_{*,p}, \alpha_{wl,*}, \alpha_{em,*}, \alpha_{sm,*} \in [0.0, .., 1.0],$$

with

$$\alpha_{*,p} + \alpha_{wl,*} + \alpha_{em,*} + \alpha_{sm,*} = 1.0$$

The results of the different weighting configurations are illustrated in Figure 6.19a. The used metric is nDCG. As conveyed by the figure, the different weightings have only a small influence on the resulting nDCG. The lowest value is 0.033 and the highest is 0.067, whereas the result without using the additional channels is 0.046. In this sense, the accuracy increased slightly when the channels are weighted as:

$$\alpha_{*,p} = 0.7, \alpha_{wl,*} = 0.1, \alpha_{em,*} = 0.1, \alpha_{sm,*} = 0.1$$

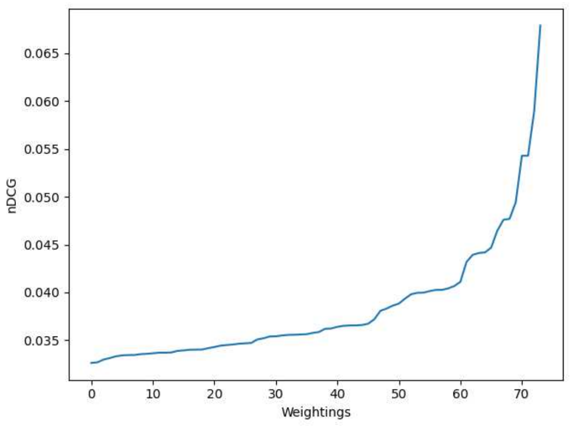
In case of *ALS*, the weighting for purchases is set as $\alpha_{*,p} = 40$ as it shows good results. The other weightings cover the following intervals:

$$\alpha_{wl,*}, \alpha_{em,*}, \alpha_{sm,*} \in [40, .., 200]$$

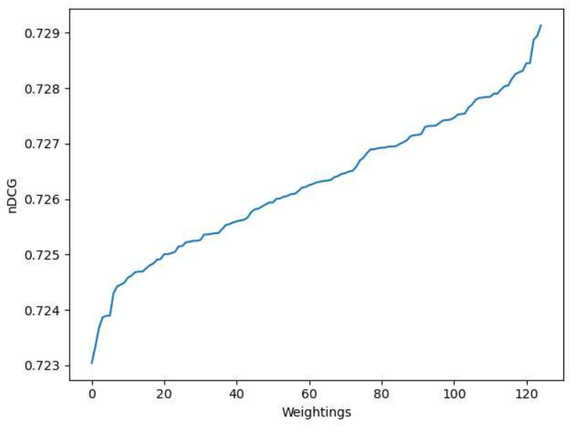
The results are shown in Figure 6.19b. To recall the result of *als_w_40*, which is based on the purchase data, is 0.73. Compared to this, only small differences are visible. Furthermore, the best result, with a value of 0.729, is achieved by the following weighting configuration:

$$\alpha_{*,p} = 40, \alpha_{wl,*} = 160, \alpha_{em,*} = 40, \alpha_{sm,*} = 120$$

However, this shows no considerable improvement in the recommendation accuracy.



(a) 75 weighting combinations applied to *II*.



(b) 125 weighting combinations applied to *ALS*.

Figure 6.19: The results of different weighting combinations sorted by the resulting nDCG values in ascending order.

Conclusion

The overall analysis of the aggregation method results reveals no considerable improvement in the recommendation accuracy by incorporating the customer signals of the channels *weblog*, *email* and *social media*. However, it must be noted that incorporating also does not influence the results negatively. This means the resulting nDCG values show a similar quality as the ones only considering the purchases of a customer, as shown in Figure 6.17b. Nevertheless, it must be clearly stated that the incorporation of the additional channels extends the number of customers and products. For instance, in *ALS*, the number of customers increased from 59,950 to 61,703 and the number of products from 5,787 to 10,502. Having this in mind, the stable accuracy results can be considered an improvement since more customers receive and more products are part of the recommendations. This alleviates the cold-start problem of customers and products. For instance, a product might be liked on social media but not purchased so far. This product would not be recommended when only the purchase data is used. The same applies to customers. When a customer did not purchase a product but clicked a newsletter of an email, the recommender system can generate appropriate recommendations; otherwise, no recommendations could be created. In this sense, from a business and customer perspective, the incorporation of additional channels is worth the effort. From a technical perspective, this incorporation leads to more extended training and testing times. Besides, the response time is negatively influenced. However, especially in the application of *ALS* the differences are relatively small and can be neglected. Consequently, this leads to the statement that incorporation is beneficial.

6.2.7 Discussion

Although incorporating the omni-channel data shows no improvement in the recommendation accuracy for the particular online retailer, companies such as Netflix, Mendeley, and Zalando collect and utilize omni-channel data in their recommender systems, as discussed and elaborated in Section 2.8. Accordingly, it seems to contradict the presented results in this thesis for those companies to invest effort in incorporating the omni-channel data.

One reason for this is that recommendations generated by these companies are mostly based on a combination of various recommender system approaches implemented as hybrid systems (see Section 2.4.3). This means, for instance, that for each user signal type in the omni-channel data, a different recommender system approach could be applied to achieve high accuracy by combining the recommendations of the approaches at the end. Besides, the parameter settings of the recommender system approaches are heavily optimized. In contrast to that, the focus of this thesis is the application of one recommender system approach at a time based on different aggregations of the omni-channel data.

Furthermore, these companies can conduct online evaluations, such as A/B testing or user studies, as discussed in Section 2.6.2. An online evaluation might show different results since the recommender system is deployed and used by the customers such that direct feedback is available, whereas the results in this thesis are based on an offline evaluation.

7 Conclusion

This section summarizes the thesis' outcome and gives an outlook about future ideas as well as challenges.

7.1 Summary

Nowadays, companies aim to provide their customers with preferably personalized products and services, and recommender systems provide customers with products they are most likely interested in. To achieve this, companies are collecting and combining as much information about their customers as possible. On the other hand, customers today leave much information revealing their interests and preferences by using different communication channels offered by companies. In the case of an online retailer, these channels might include an online shop, email communication, or a social media presence. Through these channels' usage, the customers provide insights about their interests and preferences, so-called signals. During browsing in an online catalog, these signals are, for instance, product views and searches or purchases. Engaging with a company on social media might include to like or comment on new product promotions. However, it is not directly clear how confident the gained information of these signals is. For instance, is a purchase more trustworthy than a like or comment?

This leads to the first research question of this thesis:

1. How to evaluate recommender systems based on omni-channel data?

To do so, Chapter 2 provided an overview of the various application domains of recommender systems. Consequently, the established recommender system approaches Collaborative Filtering and Content-based Filtering were introduced and explained. Then, the evaluation of recommender systems was elaborated. This included a comprehensive overview of metrics applied in the evaluation process. Additionally, available libraries and frameworks were discussed, which support this process. The chapter concluded with a detailed investigation of industrial implementations from three different application domains.

After that, Chapter 3 introduced benchmarking as a process for evaluation. Therefore, first, the historical context of benchmarking and its impact for the Information Technology was explained. Accordingly, different benchmark types and requirements were discussed. Then, state-of-the-art benchmarks covering different system aspects were investigated. This was done by considering their data models, workloads, metrics, and designs.

In Chapter 4, this knowledge led to a benchmark concept to evaluate recommender systems based on omni-channel data. Therefore, first, the idea of channels and signals in the omni-channel context was elaborated. Based on that, a data model was developed, which covers the entities for this scenario. Consequently, the concept provides an overview of data generation aspects that have to be addressed to mimic a real-world scenario. The central part of the concept was addressed to introduce data aggregation methods that define possible ways to weight and combine user signals among the channels. At this point, two approaches were introduced, a *weighted* and a *sequence-based* aggregation. As the names suggest, the former addresses to weight signals individually for each channel, for instance, if a like is more trustworthy than a click on a product in a newsletter. The latter considers user signals from a timely perspective aiming to put the highest confidence in the last user signals. For the evaluation, the concept suggests evaluating the aggregations using three scenarios: data loading, model training, and testing.

Then, the thesis continues in answering the second research question:

2. How does omni-channel data influence recommendations?

Therefore, in Chapter 5, an prototypical implementation of the benchmark concept was explained. The prototype was implemented in a modular way such that it is possible to adapt and extend it. Furthermore, it comprises the main characteristics of a benchmark. The implementation contains modules for *loading*, *preprocessing*, *aggregation*, *splitting*, *algorithms*, *configuration*, and *visualization*. The applied algorithms are implementations of established recommender system libraries. Furthermore, the chapter explained the defined settings and implemented metrics. These metrics evaluate the recommendations from a user, business, and technical perspective.

In Chapter 6 the implementation was applied to a real-world data set from a retailer. First, the chapter provided an analysis of the data. The utilized data comprised purchases from an online shop and brick and mortar stores. Additionally, views, searches from logs, and likes and comments from social media were part of the data. Besides, newsletter interactions were available as sent, opened, and clicked signals. Based on this data and the prototypical benchmark implementation, the chapter provided an extensive result analysis. This included four different applications of the benchmark based on the aggregation methods. An overall analysis was given at the end of the chapter.

Generally, the incorporation of the data showed no quality improvement regarding recommendation accuracy. However, the accuracy results also did not decrease compared to only use the purchase data. Indeed, the results reveal that their values are relatively stable and provide nearly similar results. This holds especially for the results of the applied Alternating Least Squares implementation. Due to the similarity of the accuracy, it is meaningful to consider another aspect of incorporating additional data, such as the number of customers and products. As shown by the analysis, both numbers increase through the usage of the additional channels. This means that for customers who did not purchase a product in the examined time interval but, for instance, liked a product on social media, a list of recommendations can be provided now. Besides, products, which have not been purchased so far, were not considered as possible recommendations for a customer. Now, products based on other signals become possible recommendations for the customers. From the business perspective, this is beneficial since the recommendations cover more products of their catalog. Considering these aspects, the accuracy results must be interpreted differently. In this sense, an increase in customers and products with a stable accuracy result is already an improvement.

7.2 Outlook

The introduced benchmark concept and its prototypical implementation provide a first step to evaluate recommender systems based on omni-channel data. In this sense, it is considered as a starting point that enables the exploration of further ideas in the future. Such ideas are discussed in the following.

Concept

Currently the concept contains two aggregation methods, namely, *weighted* and *sequence-based*. At this point, other approaches could be developed and integrated. One idea in this regard might integrate customer and product characteristics in the aggregation process. For instance, to use the *weighted* aggregation approaches for specific customer groups. In this regard, customers could be grouped by, e.g., demographic, social, or geospatial aspects. The proposed data model already supports user and item characteristics in the form of features.

Application Domains

In this thesis, the benchmark has been applied to channels and signals provided by a specific retailer covering the e-commerce application domain. Given the limited amount of channel data for the considered retailer, other retailers have to be examined based on their channels and signals to investigate the influence of their incorporation further and compare the results. In doing so, an overall analysis for a complete application domain might be possible. Besides, the consideration of other application domains is of interest. For instance, this means to investigate channels and signals provided in e-business, e-learning, or e-tourism.

Implementation

As already stated, the current implementation serves as a starting point. Considering the individual modules, the following extensions appear promising as future work. For instance, the integrated algorithms only work on static data sets neglecting streaming data. Therefore, algorithms that update their models incrementally have to be integrated and analyzed. This further enables to compare the algorithms by their competitive ratio.

Furthermore, the number of metrics could be extended. One exciting metric in this regard might be the Intra-List Similarity to prove a recommendation list's diversity, for instance, to avoid that a recommendation list contains too many similar products. Another idea is to compare the recommendation lists provided by the different implementations and the incorporated data sets on a user basis, for instance, to measure whether certain products are always recommended for a customer. This leads to the idea of measuring the bias and fairness of different recommender system approaches. Therefore, a metric should be part of the implementation that compares the results under this aspect.

Extending the benchmark with additional metrics only leads to insights when the results can be compared efficiently. In this sense, other ways to visualize the results are required. In this regard, one idea might be implementing radar charts, which can visualize many dimensions, in this case, metrics, in a well-arranged representation.

Bibliography

[AB15] Xavier Amatriain and Justin Basilico. Recommender systems in industry: A netflix case study. In *Recommender Systems Handbook*, pages 385–419. Springer US, Boston, MA, 2015.

[ABB$^+$85] Anon, Dina Bitton, Mark Brown, Rick Catell, Stefano Ceri, Tim Chou, Dave DeWitt, Dieter Gawlick, Hector Garcia-Molina, Bob Good, Jim Gray, Pete Homan, Bob Jolls, Tony Lukes, Ed Lazowska, John Nauman, Mike Pong, Alfred Spector, Kent Trieber, Harald Sammer, Omri Serlin, Mike Stonebraker, Andreas Reuter, and Peter Weinberger. A measure of transaction processing power. *Datamation*, 31(7):112–118, April 1985.

[AF01] Taiwo Amoo and Hershey Friedman. Do Numeric Values Influence Subjects' Responses to Rating Scales? *Journal of International Marketing and Marketing Research*, 2001.

[Aio13] Fabio Aiolli. Efficient top-n recommendation for very large scale binary rated datasets. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, page 273–280, New York, NY, USA, 2013. Association for Computing Machinery.

[Ama13] Xavier Amatriain. Big & personal: Data and models behind netflix recommendations. In *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, BigMine '13, page 1–6, New York, NY, USA, 2013. Association for Computing Machinery.

[AMBM20] Himan Abdollahpouri, Masoud Mansoury, Robin Burke, and Bamshad Mobasher. The connection between popularity bias, calibration, and fairness in recommendation. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 726–731, New York, NY, USA, 2020. Association for Computing Machinery.

[BCT18] Shlomo Berkovsky, Iván Cantador, and Domonkos Tikk. *Collaborative Recommendations.* World Scientific, 2018.

[BDT83] Dina Bitton, David J. DeWitt, and Carolyn Turbyfill. Benchmarking database systems a systematic approach. In *Proceedings of the 9th International Conference on Very Large Data Bases*, VLDB '83, page 8–19, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.

[BFCK20] Oren Barkan, Yonatan Fuchs, Avi Caciularu, and Noam Koenigstein. Explainable recommendations via attentive multi-persona collaborative filtering. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 468–473, New York, NY, USA, 2020. Association for Computing Machinery.

[BGSZ17] Sonia Bergamaschi, Luca Gagliardelli, Giovanni Simonini, and Song Zhu. Big-Bench Workload Executed by using Apache Flink. *Procedia Manufacturing*, 11:695–702, 2017.

[Bog14] Anja Bog. *Benchmarks for Transaction and Analytical Processing Systems.* Springer-Verlag, 2014.

[BOHG13] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, July 2013.

[Buc69] Werner Buchholz. A synthetic job for measuring system performance. *IBM Systems Journal*, 1969.

[Bur00] Robin Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Information Systems*, 69:175–186, 2000.

[Bur02] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, November 2002.

[Bur07] Robin Burke. *Hybrid Web Recommender Systems*, page 377–408. Springer-Verlag, Berlin, Heidelberg, 2007.

[BWT17] David Bermbach, Erik Wittern, and Stefan Tai. *Cloud Service Benchmarking: Measuring Quality of Cloud Services from a Client Perspective.* Springer Publishing Company, Incorporated, 1st edition, 2017.

[BY20] Ricardo Baeza-Yates. Bias in search and recommender systems. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 2, New York, NY, USA, 2020. Association for Computing Machinery.

[Cam89] Robert C. Camp. *Benchmarking: The search for industry best practices that lead to superior performance.* Productivity Press, 1989.

[Cat10] Rick Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Record*, 2010.

[CCFF11] Fidel Cacheda, Víctor Carneiro, Diego Fernández, and Vreixo Formoso. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Trans. Web*, 5(1), February 2011.

[CDC14] Pedro G. Campos, Fernando Díez, and Iván Cantador. Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, 24(1–2):67–119, February 2014.

[CGL+16] Paul Cao, Bhaskar Gowda, Seetha Lakshmi, Chinmayi Narasimhadevara, Patrick Nguyen, John Poelman, Meikel Poess, and Tilmann Rabl. From bigbench to tpcx-bb: Standardization of a big data benchmark. In *Performance Evaluation and Benchmarking. Traditional - Big Data - Interest of Things - 8th TPC Technology Conference, TPCTC 2016, New Delhi, India, September 5-9, 2016, Revised Selected Papers*, volume 10080 of *Lecture Notes in Computer Science*, pages 24–44. Springer, 2016.

[CHI+15a] Mihai Capotă, Tim Hegeman, Alexandru Iosup, Arnau Prat-Pérez, Orri Erling, and Peter Boncz. Graphalytics: A big data benchmark for graph-processing platforms. In *3rd International Workshop on Graph Data Management Experiences and Systems, GRADES 2015 - co-located with SIGMOD/PODS 2015*, 2015.

[CHT+17] Matthias Carnein, Leschek Homann, Heike Trautmann, Gottfried Vossen, and Karsten Kraume. Customer service in social media: An empirical study of the airline industry. In *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 6.-10. März 2017, Stuttgart, Germany, Workshopband*, volume P-266 of *LNI*, pages 33–40. GI, 2017.

[CHTV19] Matthias Carnein, Leschek Homann, Heike Trautmann, and Gottfried Vossen. A recommender system based on omni-channel customer data. In *21st IEEE Conference on Business Informatics, CBI 2019, Moscow, Russia, July 15-17, 2019, Volume 1 - Research Papers*, pages 65–74. IEEE, 2019.

[CJSD08] Max Chevalier, Christine Julien, and Chantal Soule-Dupuy. *Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling*. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 1st edition, 2008.

[CLA+03] Dan Cosley, Shyong K. Lam, István Albert, Joseph A. Konstan, and John Riedl. Is seeing believing?: how recommender system interfaces affect users' opinions. In *Proceedings of the 2003 Conference on Human Factors in Computing Systems, CHI 2003, Ft. Lauderdale, Florida, USA, April 5-10, 2003*, pages 585–592. ACM, 2003.

[CLEM11] Badrish Chandramouli, Justin J. Levandoski, Ahmed Eldawy, and Mohamed F. Mokbel. Streamrec: A real-time recommender system. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, page 1243–1246, New York, NY, USA, 2011. Association for Computing Machinery.

[CRS+13] Badrul Chowdhury, Tilmann Rabl, Pooya Saadatpanah, Jiang Du, and Hans-Arno Jacobsen. A bigbench implementation in the hadoop ecosystem. In *Advancing Big Data Benchmarks - Proceedings of the 2013 Workshop Series on Big Data Benchmarking, WBDB.cn, Xi'an, China, July 16-17, 2013 and WBDB.us,* *San José, CA, USA, October 9-10, 2013 Revised Selected Papers*, volume 8585 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2013.

[CSS$^+$18] Diego Carvalho, Nícollas Silva, Thiago Silveira, Fernando Mourão, Adriano C. M. Pereira, Diego Dias, and Leonardo C. da Rocha. Fair: A framework for analyses and evaluations on recommender systems. In *Computational Science and Its Applications - ICCSA 2018 - 18th International Conference, Melbourne, VIC, Australia, July 2-5, 2018, Proceedings, Part III*, volume 10962 of *Lecture Notes in Computer Science*, pages 383–397. Springer, 2018.

[CST$^+$10] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, page 143–154, New York, NY, USA, 2010. Association for Computing Machinery.

[CTP$^+$20] Konstantina Christakopoulou, Madeleine Traverse, Trevor Potter, Emma Marriott, Daniel Li, Chris Haulk, Ed H. Chi, and Minmin Chen. Deconfounding user satisfaction estimation from response rate bias. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 450–455, New York, NY, USA, 2020. Association for Computing Machinery.

[DCJ19] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 101–109, New York, NY, USA, 2019. Association for Computing Machinery.

[DDGR07] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 271–280, New York, NY, USA, 2007. Association for Computing Machinery.

[DK04] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, January 2004.

[Eks18] Michael D. Ekstrand. The lkpy package for recommender systems experiments. Computer Science Faculty Publications and Presentations 147, Boise State University, Aug 2018.

[Emr08] Christian Emrich. *Multi-Channel-Communications- und Marketing-Management.* Gabler Verlag, 2008.

[EY36] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1936.

[FB08] A. Felfernig and R. Burke. Constraint-based recommender systems: Technologies and research issues. In *Proceedings of the 10th International Conference* *on Electronic Commerce*, ICEC '08, New York, NY, USA, 2008. Association for Computing Machinery.

[FMY+19] Wenqi Fan, Yao Ma, Dawei Yin, Jianping Wang, Jiliang Tang, and Qing Li. Deep social collaborative filtering. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 305–313, New York, NY, USA, 2019. Association for Computing Machinery.

[FPK+17] Erzsébet Frigó, Róbert Pálovics, Domokos Kelen, Levente Kocsis, and András A. Benczúr. Alpenglow: Open source recommender framework with time-aware learning and evaluation. In *Proceedings of the Poster Track of the 11th ACM Conference on Recommender Systems (RecSys 2017), Como, Italy, August 28, 2017*, volume 1905 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[Fre17] Antonino Freno. Practical lessons from developing a large-scale recommender system at zalando. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 251–259, New York, NY, USA, 2017. Association for Computing Machinery.

[GBB16] Benjamin Gras, Armelle Brun, and Anne Boyer. Identifying grey sheep users in collaborative filtering: A distribution-based technique. In *Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization*, UMAP '16, page 17–26, New York, NY, USA, 2016. Association for Computing Machinery.

[GBCV11] Cristina Gena, Roberto Brogi, Federica Cena, and Fabiana Vernero. The impact of rating scales on user's rating behavior. In *Proceedings of the 19th International Conference on User Modeling, Adaption, and Personalization*, UMAP'11, page 123–134, Berlin, Heidelberg, 2011. Springer-Verlag.

[GM05] Thomas George and Srujana Merugu. A scalable collaborative filtering framework based on co-clustering. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, page 625–628, USA, 2005. IEEE Computer Society.

[Gra93] Jim Gray. The Benchmark Handbook for Database and Transaction Systems. *The Benchmark Handbook for Database and Transaction Systems*, 1993.

[GRFST11] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Mymedialite: A free recommender system library. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, page 305–308, New York, NY, USA, 2011. Association for Computing Machinery.

[GUH16] Carlos A. Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4), December 2016.

[Guy15] Ido Guy. Social recommender systems. In *Recommender Systems Handbook, Second Edition.* 2015.

[HCZ$^+$15] Yanxiang Huang, Bin Cui, Wenyu Zhang, Jie Jiang, and Ying Xu. TencentRec: Real-time Stream Recommendation in Practice. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 227–238, New York, NY, USA, 2015. ACM.

[HJ65] R. F. Hitti and E. O. Joslin. Session 2: Evaluation and performance of computers: 2.1: Application benchmarks: The key to meaningful computer evaluations. In *Proceedings of the 1965 20th National Conference, ACM 1965*, 1965.

[HJZ18] Rui Han, Lizy Kurian John, and Jianfeng Zhan. Benchmarking big data systems: A review. *IEEE Transactions on Services Computing*, 11(3):580–597, 2018.

[HKTR04] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004.

[HKV08] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, page 263–272, USA, 2008. IEEE Computer Society.

[HLX14] Rui Han, Xiaoyi Lu, and Jiangtao Xu. On big data benchmarking. In Jianfeng Zhan, Rui Han, and Chuliang Weng, editors, *Big Data Benchmarks, Performance Optimization, and Emerging Hardware - 4th and 5th Workshops, BPOE 2014, Salt Lake City, USA, March 1, 2014 and Hangzhou, China, September 5, 2014, Revised Selected Papers*, volume 8807 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2014.

[HOdRvH20] Jin Huang, Harrie Oosterhuis, Maarten de Rijke, and Herke van Hoof. Keeping dataset biases out of the simulation: A debiased simulator for reinforcement learning based recommender systems. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 190–199, New York, NY, USA, 2020. Association for Computing Machinery.

[Hol14] Heinrich Holland. *Digitales Dialogmarketing.* Gabler Verlag, 2014.

[HPV16] Chen He, Denis Parra, and Katrien Verbert. Interactive recommender systems: A survey of the state of the art and future research challenges and opportunities. *Expert Systems with Applications*, 2016.

[Hup09] Karl Huppler. The art of building a good benchmark. In *Performance Evaluation and Benchmarking: First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*, page 18–30, Berlin, Heidelberg, 2009. Springer-Verlag.

[HZS+15] Rui Han, Shulin Zhan, Chenrong Shao, Junwei Wang, Lizy K. John, Jiangtao Xu, Gang Lu, and Lei Wang. Bigdatabench-mt: A benchmark tool for generating realistic mixed data center workloads. In *Big Data Benchmarks, Performance Optimization, and Emerging Hardware - 6th Workshop, BPOE 2015, Kohala, HI, USA, August 31 - September 4, 2015. Revised Selected Papers*, volume 9495 of *Lecture Notes in Computer Science*, pages 10–21. Springer, 2015.

[IBGZ18] Todor Ivanov, Patrick Bedué, Ahmad Ghazal, and Roberto V. Zicari. Adding velocity to bigbench. In *Proceedings of the Workshop on Testing Database Systems*, DBTest'18, New York, NY, USA, 2018. Association for Computing Machinery.

[IHN+16] Alexandru Iosup, Tim Hegeman, Wing Lung Ngai, Stijn Heldens, Arnau Prat-Pérez, Thomas Manhardto, Hassan Chafio, Mihai Capotă, Narayanan Sundaram, Michael Anderson, Ilie Gabriel Tănase, Yinglong Xia, Lifeng Nai, and Peter Boncz. Ldbc graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms. *Proc. VLDB Endow.*, 9(13):1317–1328, September 2016.

[IRP+15] Todor Ivanov, Tilmann Rabl, Meikel Poess, Anna Queralt, John Poelman, Nicolás Poggi, and Jeffrey Buell. Big data benchmark compendium. In *Performance Evaluation and Benchmarking: Traditional to Big Data to Internet of Things - 7th TPC Technology Conference, TPCTC 2015, Kohala Coast, HI, USA, August 31 - September 4, 2015. Revised Selected Papers*, volume 9508 of *Lecture Notes in Computer Science*, pages 135–155. Springer, 2015.

[IS18] Todor Ivanov and Rekha Singhal. ABench: Big data architecture stack benchmark. In *ICPE 2018 - Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018.

[JJK18] Michael Jugovac, Dietmar Jannach, and Mozhgan Karimi. Streamingrec: A framework for benchmarking stream-based news recommenders. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, page 269–273, New York, NY, USA, 2018. Association for Computing Machinery.

[JM15] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects, 2015.

[JMO19] Amir H. Jadidinejad, Craig Macdonald, and Iadh Ounis. Unifying explicit and implicit feedback for rating prediction and ranking recommendation tasks. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '19, page 149–156, New York, NY, USA, 2019. Association for Computing Machinery.

[JVG19] Olivier Jeunen, Koen Verstrepen, and Bart Goethals. Efficient similarity computation for collaborative filtering in dynamic environments. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 251–259, New York, NY, USA, 2019. Association for Computing Machinery.

[JZFF10] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems: An introduction.* Cambridge University Press, 2010.

[KB16] Marius Kaminskas and Derek Bridge. Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM Trans. Interact. Intell. Syst.*, 7(1), December 2016.

[KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.

[KH17] Alexandros Karatzoglou and Balázs Hidasi. Deep learning for recommender systems. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 396–397, New York, NY, USA, 2017. Association for Computing Machinery.

[KJJ18] Mozhgan Karimi, Dietmar Jannach, and Michael Jugovac. News recommender systems – Survey and roads ahead. *Information Processing and Management*, 2018.

[KWV16] Denis Kotkov, Shuaiqiang Wang, and Jari Veijalainen. A survey of serendipity in recommender systems. *Know.-Based Syst.*, 111(C):180–192, November 2016.

[Lan01] Doug Laney. 3D Data Management: Controlling Data Volume, Velocity, and Variety. *Application Delivery Strategies*, 2001.

[LC85] Bryon C. Lewis and Albert E. Crews. The evolution of benchmarking as a computer performance evaluation technique. *MIS Q.*, 9(1):7–16, March 1985.

[LH16] Philip Lenhart and Daniel Herzog. Combining content-based and collaborative filtering for personalized sports news recommendations. In *Proceedings of the 3rd Workshop on New Trends in Content-Based Recommender Systems co-located with ACM Conference on Recommender Systems (RecSys 2016), Boston, MA, USA, September 16, 2016*, volume 1673 of *CEUR Workshop Proceedings*, pages 3–10. CEUR-WS.org, 2016.

[Liu09] Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data.* Springer-Verlag, Berlin, Heidelberg, 2009.

[LSY03] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003.

[LTW$^+$17] Min Li, Jian Tan, Yandong Wang, Li Zhang, and Valentina Salapura. Sparkbench: A spark benchmarking suite characterizing large-scale in-memory data analytics. *Cluster Computing*, 20(3):2575–2589, September 2017.

[LWM$^+$15] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. Recommender system application developments. *Decis. Support Syst.*, 74(C):12–32, June 2015.

[LWXH14] Ruirui Lu, Gang Wu, Bin Xie, and Jingtong Hu. Stream bench: Towards benchmarking modern distributed stream computing frameworks. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, UCC '14, page 69–78, USA, 2014. IEEE Computer Society.

[MTW$^+$20] Peter Mattson, Hanlin Tang, Gu-Yeon Wei, Carole-Jean Wu, Vijay Janapa Reddi, Christine Cheng, Cody Coleman, Greg Diamos, David Kanter, Paulius Micikevicius, David A. Patterson, and Guenther Schmuelling. Mlperf: An industry standard benchmark suite for machine learning performance. *IEEE Micro*, 40(2):8–16, 2020.

[MZS20] Alessandro B. Melchiorre, Eva Zangerle, and Markus Schedl. Personality bias of music recommendation algorithms. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 533–538, New York, NY, USA, 2020. Association for Computing Machinery.

[NLW$^+$09] Raghunath Othayoth Nambiar, Matthew Lanken, Nicholas Wakou, Forrest Carman, and Michael Majdalany. Transaction processing performance council (tpc): Twenty years later — a look back, a look ahead. In *Performance Evaluation and Benchmarking: First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*, page 1–10, Berlin, Heidelberg, 2009. Springer-Verlag.

[PB07] Michael J. Pazzani and Daniel Billsus. *Content-Based Recommendation Systems*, volume 4321 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2007.

[PKCK12] Deuk Hee Park, Hyea Kyeong Kim, Il Young Choi, and Jae Kyeong Kim. A literature review and classification of recommender systems research. *Expert Syst. Appl.*, 39(11):10059–10072, September 2012.

[PRJ17] Meikel Poess, Tilmann Rabl, and Hans-Arno Jacobsen. Analysis of tpc-ds: The first standard benchmark for sql-based big data systems. In *Proceedings of the 2017 Symposium on Cloud Computing*, page 573–585, New York, NY, USA, 2017. Association for Computing Machinery.

[QCJ18] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. Sequence-aware recommender systems. *ACM Comput. Surv.*, 51(4), July 2018.

[Raa19] Francois Raab. *System Under Test.* Springer, 2019.

[RCK$^+$20] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark

Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. Mlperf inference benchmark. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, page 446–459. IEEE Press, 2020.

[RFD$^+$14] Tilmann Rabl, Michael Frank, Manuel Danisch, Bhaskar Gowda, and Hans-Arno Jacobsen. Towards a complete bigbench implementation. In *Big Data Benchmarking - 5th International Workshop, WBDB 2014, Potsdam, Germany, August 5-6, 2014, Revised Selected Papers*, volume 8991 of *Lecture Notes in Computer Science*, pages 3–11. Springer, 2014.

[RFGST09] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, UAI 2009*, 2009.

[RGH$^+$12] Tilmann Rabl, Ahmad Ghazal, Minqing Hu, Alain Crolotte, Francois Raab, Meikel Poess, and Hans-Arno Jacobsen. Bigbench specification v0.1. In *Revised Selected Papers of the First Workshop on Specifying Big Data Benchmarks - Volume 8163*, page 164–201, Berlin, Heidelberg, 2012. Springer-Verlag.

[RMWZ14] Martin P. Robillard, Walid Maalej, Robert J. Walker, and Thomas Zimmermann. *Recommendation Systems in Software Engineering.* Springer Publishing Company, Incorporated, 2014.

[Rol95] Asbjørn Rolstadås. *Benchmarking — Theory and Practice.* Springer US, 1 edition, 1995.

[RRS11] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to Recommender Systems Handbook. In *Recommender Systems Handbook*. 2011.

[RSP20] R. Ramanathan, Nicolas K. Shinada, and Sucheendra K. Palaniappan. Building a reciprocal recommendation system at scale from scratch: Learnings from one of japan's prominent dating applications. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 566–567, New York, NY, USA, 2020. Association for Computing Machinery.

[SB14a] Alan Said and Alejandro Bellogín. Comparative recommender system evaluation: Benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, page 129–136, New York, NY, USA, 2014. Association for Computing Machinery.

[SB14b] Alan Said and Alejandro Bellogín. Rival: A toolkit to foster reproducibility in recommender system evaluation. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, page 371–372, New York, NY, USA, 2014. Association for Computing Machinery.

[Ser91] Omri Serlin. The history of debitcredit and the TPC. In Jim Gray, editor, *The Benchmark Handbook for Database and Transaction Systems (1st Edition)*, pages 19–38. Morgan Kaufmann, 1991.

[Sha48] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 1948.

[SKKR01] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 285–295, New York, NY, USA, 2001. Association for Computing Machinery.

[SLK$^+$16] Mario Scriminaci, Andreas Lommatzsch, Benjamin Kille, Frank Hopfgartner, Martha A. Larson, Davide Malagoli, András Serény, and Till Plumbaum. Idomaar: A framework for multi-dimensional benchmarking of recommender algorithms. In *Proceedings of the Poster Track of the 10th ACM Conference on Recommender Systems (RecSys 2016), Boston, USA, September 17, 2016*, volume 1688 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[SPBD19] Abhaya Kumar Sahoo, Chittaranjan Pradhan, Rabindra Kumar Barik, and Harishchandra Dubey. Deepreco: Deep learning based health recommender system using collaborative filtering. *Computation*, 7(2):25, 2019.

[Sta09] Tim Stapenhurst. *The Benchmarking Book*. Taylor & Francis Ltd., 2009.

[Ste18] Harald Steck. Calibrated recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, page 154–162, New York, NY, USA, 2018. Association for Computing Machinery.

[STS$^+$12] Alan Said, Domonkos Tikk, Klara Stumpf, Yue Shi, Martha A. Larson, and Paolo Cremonesi. Recommender systems evaluation: A 3d benchmark. In *Proceedings of the Workshop on Recommendation Utility Evaluation: Beyond RMSE, RUE 2012, Dublin, Ireland, September 9, 2012*, volume 910 of *CEUR Workshop Proceedings*, pages 21–23. CEUR-WS.org, 2012.

[STSO20] Masahiro Sato, Sho Takemori, Janmajay Singh, and Tomoko Ohkuma. Unbiased learning for the causal effect of recommendation. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 378–387, New York, NY, USA, 2020. Association for Computing Machinery.

[TDD$^+$17] Xinhui Tian, Shaopeng Dai, Zhihui Du, Wanling Gao, Rui Ren, Yaodong Cheng, Zhifei Zhang, Zhen Jia, Peijian Wang, and Jianfeng Zhan. Bigdatabench-s: An open-source scientific big data benchmark suite. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPS Workshops 2017, Orlando / Buena Vista, FL, USA, May 29 - June 2, 2017*, pages 1068–1077. IEEE Computer Society, 2017.

[TG20] Kosetsu Tsukuda and Masataka Goto. Explainable recommendation for repeat consumption. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 462–467, New York, NY, USA, 2020. Association for Computing Machinery.

[TPNT09] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, June 2009.

[Tuc00] Friedrich W. Frhr Tucher. *Benchmarking von Wissensmanagement.* Deutscher Universitätsverlag, 2000.

[VG15] Koen Verstrepen and Bart Goethals. Top-n recommendation for shared accounts. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, page 59–66, New York, NY, USA, 2015. Association for Computing Machinery.

[VM09] Marco Vieira and Henrique Madeira. From performance to dependability benchmarking: A mandatory path. In *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*, volume 5895 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2009.

[VZV$^+$16] André Calero Valdez, Martina Ziefle, Katrien Verbert, Alexander Felfernig, and Andreas Holzinger. Recommender systems for health informatics: State-of-the-art and future perspectives. In *Machine Learning for Health Informatics - State-of-the-Art and Future Challenges*, volume 9605 of *Lecture Notes in Computer Science*, pages 391–414. Springer, 2016.

[WBCR17] Kebing Wang, Bianny Bian, Paul Cao, and Mike Riess. Experiences and lessons in practice using tpcx-bb benchmarks. In *Performance Evaluation and Benchmarking for the Analytics Era - 9th TPC Technology Conference, TPCTC 2017, Munich, Germany, August 28, 2017, Revised Selected Papers*, volume 10661 of *Lecture Notes in Computer Science*, pages 93–102. Springer, 2017.

[WCZ$^+$16] Jian Wei, Kai Chen, Yi Zhou, Qu Zhou, and Jianhua He. Benchmarking of distributed computing engines spark and graphlab for big data analytics. In *Second IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2016, Oxford, United Kingdom, March 29 - April 1, 2016*, pages 10–13. IEEE Computer Society, 2016.

[Weg03] Ingo Wegener. *Komplexitätstheorie - Grenzen der Effizienz von Algorithmen.* Springer, 2003.

[Wei90] Reinhold P. Weicker. An overview of common benchmarks. *Computer*, 23(12):65–75, December 1990.

[WM18] Mengting Wan and Julian McAuley. Item recommendation on monotonic behavior chains. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, page 86–94, New York, NY, USA, 2018. Association for Computing Machinery.

[WP14] Martin Wiesner and Daniel Pfeifer. Health recommender systems: Concepts, requirements, technical basics and challenges. *International Journal of Environmental Research and Public Health*, 2014.

[WZL$^+$14] Lei Wang, Jianfeng Zhan, Chunjie Luo, Yuqing Zhu, Qiang Yang, Yongqiang He, Wanling Gao, Zhen Jia, Yingjie Shi, Shujie Zhang, Chen Zheng, Gang Lu, Kent Zhan, Xiaona Li, and Bizhu Qiu. Bigdatabench: A big data benchmark suite from internet services. In *20th IEEE International Symposium on High Performance Computer Architecture, HPCA 2014, Orlando, FL, USA, February 15-19, 2014*, pages 488–499. IEEE Computer Society, 2014.

[XYE$^+$16] Wen Xiong, Zhibin Yu, Lieven Eeckhout, Zhengdong Bei, Fan Zhang, and Cheng-Zhong Xu. Shenzhen transportation system (SZTS): a novel big data benchmark suite. *Journal of Supercomputing*, 72(11):4337–4364, 2016.

[YBG$^+$18] Longqi Yang, Eugene Bagdasaryan, Joshua Gruenstein, Cheng-Kang Hsieh, and Deborah Estrin. Openrec: A modular framework for extensible and adaptable recommendation algorithms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 664–672, New York, NY, USA, 2018. Association for Computing Machinery.

[YLH$^+$20] Bowen Yuan, Yaxu Liu, Jui-Yang Hsia, Zhenhua Dong, and Chih-Jen Lin. Unbiased ad click prediction for position-aware advertising systems. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 368–377, New York, NY, USA, 2020. Association for Computing Machinery.

[YMJ$^+$16] Tong Yu, Ole J. Mengshoel, Alvin Jude, Eugen Feller, Julien Forgeat, and Nimish Radia. Incremental learning for matrix factorization in recommender systems. In James Joshi, George Karypis, Ling Liu, Xiaohua Hu, Ronay Ak, Yinglong Xia, Weijia Xu, Aki-Hiro Sato, Sudarsan Rachuri, Lyle H. Ungar, Philip S. Yu, Rama Govindaraju, and Toyotaro Suzumura, editors, *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*, pages 1056–1063. IEEE Computer Society, 2016.

[ZHZC20] Ziwei Zhu, Yun He, Yin Zhang, and James Caverlee. Unbiased implicit recommendation and propensity estimation via combinational joint learning. In *Fourteenth ACM Conference on Recommender Systems*, RecSys '20, page 551–556, New York, NY, USA, 2020. Association for Computing Machinery.

[ZYST19] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52(1), February 2019.

List of Web Pages

[1] Benchcouncil: International open benchmark council. https://www.benchcouncil.org/mission.html. Last accessed: 2020-12-09.

[2] Bigdatabench - download. https://www.benchcouncil.org/BigDataBench/download.html. Last accessed: 2020-12-09.

[3] Bigdatabench - summary. https://www.benchcouncil.org/BigDataBench/index.html. Last accessed: 2020-12-09.

[4] Facebook marketplace. https://de-de.facebook.com/marketplace/. Last accessed: 2020-12-09.

[5] Mlperf benchmark suite. https://github.com/mlperf. Last accessed: 2020-12-09.

[6] Most popular online dating apps in the united states as of september 2019, by reach. https://www.statista.com/statistics/826782/most-popular-dating-apps-by-reach-usa. Last accessed: 2020-12-09.

[7] Mymedialite: Item recommendation tool. http://www.mymedialite.net/documentation/item_prediction.html. Last accessed: 2020-12-09.

[8] Mymedialite: Rating prediction tool. http://www.mymedialite.net/documentation/rating_prediction.html. Last accessed: 2020-12-09.

[9] Number of active zalando customers from 1st quarter 2014 to 4th quarter 2019. https://www.statista.com/statistics/370657/zalando-active-buyers. Last accessed: 2020-12-09.

[10] Number of monthly active facebook users worldwide as of 1st quarter 2020. https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide. Last accessed: 2020-12-09.

[11] Number of netflix paying streaming subscribers worldwide from 3rd quarter 2011 to 1st quarter 2020. https://www.statista.com/statistics/250934/quarterly-number-of-netflix-streaming-subscribers-worldwide. Last accessed: 2020-12-09.

[12] Number of online dating users in the united states from 2017 to 2024. https://www.statista.com/statistics/417654/us-online-dating-user-numbers. Last accessed: 2020-12-09.

[13] Numbers of linkedin members from 1st quarter 2009 to 3rd quarter 2016. https://www.statista.com/statistics/274050/quarterly-numbers-of-linkedin-members. Last accessed: 2020-12-09.

[14] Oryx 2. http://oryx.io/. Last accessed: 2020-12-09.

[15] Sparkbench. https://codait.github.io/spark-bench/. Last accessed: 2020-12-09.

[16] Sparkbench - adding new workloads. https://codait.github.io/spark-bench/developers-guide/adding-new-workloads/. Last accessed: 2020-12-09.

[17] Standard performance evaluation corporation. https://www.spec.org/. Last accessed: 2020-12-09.

[18] Szts - download. http://cloud.siat.ac.cn/cloud/szts/szts-download.php.

[19] Tpcx-bb tools download. http://tpc.org/TPC_Documents_Current_Versions/download_programs/tools-download-request5.asp?bm_type=TPCX-BB&bm_vers=1.4.0&mode=CURRENT-ONLY. Last accessed: 2020-12-09.

[20] What is the froomle platform? https://docs.froomle.com. Last accessed: 2020-12-09.

[21] What is the tpc. http://www.tpc.org/information/about/about.asp? Last accessed: 2020-12-09.

[22] Why ldbc. http://ldbcouncil.org/public/why-ldbc. Last accessed: 2020-12-09.

[23] Xing. https://www.xing.de. Last accessed: 2020-06-08.

[24] Ycsb - download. https://github.com/brianfrankcooper/YCSB. Last accessed: 2020-12-09.

[25] Netflix prize. https://www.netflixprize.com/, 2009. Last accessed: 2020-12-09.

[26] Evolution of the netflix data pipeline. https://netflixtechblog.com/evolution-of-the-netflix-data-pipeline-da246ca36905, 2016. Last accessed: 2020-12-09.

[27] Ldbc snb data generator. https://github.com/ldbc/ldbc_snb_datagen, 2020. Last accessed: 2020-12-09.

[28] Sparkbench - download. https://github.com/CODAIT/spark-bench, 2020. Last accessed: 2020-12-09.

[29] Tpc benchmark ™ ds - standard specification. http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf, 2020. Last accessed: 2020-12-09.

[30] Tpc download current. http://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp, 2020. Last accessed: 2020-12-09.

[31] Tpc-ds is a decision support benchmark. http://www.tpc.org/tpcds/, 2020. Last accessed: 2020-12-09.

[32] Tpc express big bench - standard specification. http://www.tpc.org/tpc_documents_current_versions/pdf/tpcx-bb_v1.4.0.pdf, 2020. Last accessed: 2020-12-09.

[33] Xing - facts & figures. https://www.new-work.se/en/about-new-work-se/facts-and-figures, 2020. Last accessed: 2020-12-09.

[34] Renzo Angles. Benchmark principles and methods. http://ldbcouncil.org/sites/default/files/LDBC_D1.1.2.pdf, 2013. Last accessed:.

[35] Alice Atkinson-Bonasio. Ten years of mendeley – and what's next. https://www.elsevier.com/connect/ten-years-of-mendeley-and-whats-next, 2018. Last accessed: 2020-12-09.

[36] Nachiket Bhat. Trivago business model – everything you need to know about how trivago works & revenue analysis. https://www.ncrypted.net/blog/trivago-business-model-everything-you-need-to-know-about-how-trivago-works-revenue-analysis, 2018. Last accessed: 2020-12-09.

[37] Sean Busbey. Running a workload. https://github.com/brianfrankcooper/YCSB/wiki/Running-a-Workload, 2019. Last accessed: 2020-12-09.

[38] Graph 500 Steering Committee. Benchmark specification. https://graph500.org/?page_id=12, 2017. Last accessed: 2020-12-09.

[39] Andrew Gaft. Travel and tourism statistics: The ultimate collection. https://blog.accessdevelopment.com/tourism-and-travel-statistics-the-ultimate-collection, 2019. Last accessed: 2020-12-09.

[40] Leschek Homann. Postgrenosql. https://github.com/brianfrankcooper/YCSB/tree/master/postgrenosql, 2018. Last accessed: 2020-12-09.

[41] Cameron Johnson. Goodbye stars, hello thumbs. https://about.netflix.com/en/news/goodbye-stars-hello-thumbs, 2017. Last accessed: 2020-12-09.

[42] Juozas Kaziukenas. Amazon replaces "reviews" with "ratings". https://www.marketplacepulse.com/articles/amazon-replaces-reviews-with-ratings, 2019. Last accessed: 2020-12-09.

[43] Takuya Kitazawa. Flurs: A python library for online item recommendation. https://takuti.me/note/flurs, 2017. Last accessed: 2020-12-09.

[44] Ed Ingold Kris Jack and Maya Hristakeva. The components of a recommender system. https://buildingrecommenders.wordpress.com/2015/11/10/the-components-of-a-recommender-system, 2015. Last accessed: 2020-12-09.

[45] Ed Ingold Kris Jack and Maya Hristakeva. Mendeley suggest architecture. https://buildingrecommenders.wordpress.com/2016/10/10/mendeley-suggest-architecture, 2016. Last accessed: 2020-12-09.

[46] Charles Levine. Tpc-c: The oltp benchmark. http://www.tpc.org/information/sessions/sigmod/sld002.htm, 1997. Last accessed: 2020-12-09.

[47] Steve Liu. https://www.slideshare.net/sessionsevents/dr-steve-liu-chief-scientist-tinder-at-mlconf-sf-2017. https://www.elsevier.com/connect/ten-years-of-mendeley-and-whats-next, 2017. Last accessed: 2020-12-09.

[48] Mendeley. https://www.mendeley.com. Last accessed: 2020-06-08.

[49] Prof. Dr. Gerald Oeser. Omni-channel-management. https://wirtschaftslexikon.gabler.de/definition/omni-channel-management-54201, 1999. Last accessed: 2020-12-09.

[50] Prasanna Padmanabhan and Roopa Tangirala. Netflix recommendations using spark + cassandra. https://www.slideshare.net/DataStax/netflix-recommendations-using-spark-cassandra, 2016. Last accessed: 2020-12-09.

[51] Jesse Steinweg-Woods. A gentle introduction to recommender systems with implicit feedback. https://jessesw.com/Rec-System/, 2016. Last accessed: 2020-12-09.

[52] Klaus Wübbenhorst. Definition: Was ist "benchmarking"? `https://wirtschaftslexikon.gabler.de/definition/benchmarking-29988`, 2020. Last accessed: 2020-12-09.

[53] Justin Basilico Xavier Amatriain. System architectures for personalization and recommendation. `https://netflixtechblog.com/system-architectures-for-personalization-and-recommendation-e081aa94b5d8`, 2013. Last accessed: 2020-12-09.

List of Abbreviations and Acronyms

$CatCov_a$	absolute CatCov.
$CatCov_r$	relative CatCov.
M_{Con}	Memory Consumption.
T_{Load}	Loading Time.
T_{Res}	Response Time.
T_{Test}	Testing Time.
T_{Train}	Training Time.
ACID	Atomicity, Consistency, Isolation, and Durability.
AI	Artificial Intelligence.
ALS	Alternating Least Squares.
API	Application Programming Interface.
ATM	Automated Teller Machine.
AUC	Area Under the Curve.
AWS	Amazon Web Services.
BASE	Basically Available, Soft state, and Eventual consistency.
BDGS	Big Data Generator Suite.
BFS	Breadth-first Search.
BPR	Bayesian Personalized Ranking.
CARS	Context-aware Recommender Systems.
CatCov	Catlog Coverage.
CB	Content-based Filtering.

CF	Collaborative Filtering.
CoC	Co-Clustering.
CPI	Cycles per Instruction.
CPU	Central Processing Unit.
CR	Conversion Rate.
CRM	Customer Relationship Management.
CSV	Comma-separated Values.
CTR	Click-through Rate.
DASE	Data Preparator, Algorithm, Serving, and Evaluation Metrics.
DBMS	Database Management System.
DCG	Discounted Cumulative Gain.
DL	Deep Learning.
EDP	Electronic Data Processing.
EPS	Edges per Second.
ERM	Entity Relationship Model.
ETL	Extract, Transform, and Load.
EVPS	Edges and Vertices per Second.
FCP	Fraction of Concordant Pairs.
G2B	Government-to-Business.
G2C	Government-to-Citizen.
GPFM	Gaussian Process Factorization Machines.
GPS	Global Positioning System.
GPU	Graphics Processing Unit.
HDFS	Hadoop Distributed File System.
HR	Hit Rate.

HTTP Hypertext Transfer Protocol.

IDF Inverse Document Frequency.

II Item-based CF.

ILS Intra-List Similarity.

IO InputOutput.

IPC Instructions per Cycle.

IR Information Retrieval.

IT Information Technology.

JSON JavaScript Object Notation.

KBRS Knowledge-based Recommender Systems.

kNN k-Nearest Neighbor.

LDBC Linked Data Benchmark Council.

LLCMPKI Last-level Cache Misses per Thousand Instructions.

LMF Logistic Matrix Factorization.

LPF Latency Penalty Factor.

LR Logistic Regression.

MAC Media Access Control.

MAE Mean Absolute Error.

MAI Memory Accesses per Instruction.

MAP Mean Average Precision.

MDCR Multi-Dimensional Collaborative Recommendation.

MF Matrix Factorization.

MIPS Million Instructions per Second.

ML Machine Learning.

MOI Map Output/Input Ratio.

MOOC Massive Open Online Courses.

MPKI Misses per Kilo Instructions.

MPR Mean Percentile Rank.

MRR Mean Reciprocal Rank.

MSE Mean Square Error.

nDCG normalized Discounted Cumulative Gain.

NLP Natural Language Processing.

NLTK Natural Language Processing Toolkit.

NMF Non-negative Matrix Factorization.

NoSQL Not only SQL.

OLAP Online Analytical Processing.

OLTP Online Transaction Processing.

PDGF Parallel Data Generator Framework.

PMF Probabilistic Matrix Factorization.

POP Popularity.

RAN Random.

RDBMS Relational Database Management System.

RDD Resilient Distributed Dataset.

RDF Resource Description Framework.

REST Representational State Transfer.

RMSE Root Mean Square Error.

RR Reciprocal Rank.

SBRS Sequence-based Recommender Systems.

SFTP Secure File Transfer Protocol.

SGD Stochastic Gradient Descent.

SkNN	session-based kNN.
SNB	Social Network Benchmark.
SPEC	Standard Performance Evaluation Corporation.
SQL	Structured Query Language.
SUT	System under Test.
SVD	Singular Value Decomposition.
SVM	Support Vector Machine.
SZTS	ShenZhen Transportation System.
TARS	Time-aware Recommender Systems.
TF	Term Frequency.
TF-IDF	Term Frequency - Inverse Document Frequency.
TMRS	Time Map/Reduce Stage Ratio.
TPC	Transaction Processing Performance Council.
TPF	Throughput Penalty Factor.
WiFi	Wireless Fidelity.
XML	Extensible Markup Language.
YAML	YAML Ain’t Markup Language.
YCSB	Yahoo! Cloud Serving Benchmark.