Master thesis

Daniel Figia

Recommendation engine: Collaborative filtering based on customer behavior on an e-commerce website

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik
Department Computer Science

Hamburg University of Applied Sciences
Faculty of Engineering and Computer Science
Department Computer Science

Daniel Figia

Recommendation engine: Collaborative filtering based on customer behavior on an e-commerce website

Master thesis
Department of Computer Science
Faculty of Engineering and Computer Science
University of Applied Sciences Hamburg

First examiner Prof. Dr. Ulrike Steffens
Second examiner: Prof. Dr. Olaf Zukunft

Submitted on: 05. January 2023

Daniel Figia

Title of Thesis

Recommendation engine: Collaborative filtering based on customer behavior on an e-commerce website

Keywords

Recommendation Engine, Collaborative Filtering, Customer behavior, E-commerce

Abstract

This thesis examined how ratings can be derived from clickstream data to generate good recommendations by using it as an input for collaborative filtering. More and more people buy their products online on an e-commerce platform. Consequently, product recommendations on these platforms are becoming increasingly important for the success of a company as better recommendations increase the chances of sales. Collaborative filtering can be used to generate recommendations. It tries to find similar customers/products for each customer/product based on customer ratings. Ratings can be given directly by a rating system or derived indirectly, for example, using clickstream data. Subsequently, the ratings can be used as an input for user-based and item-based collaborative filtering to generate recommendations. This thesis demonstrates this with clickstream data provided by novomind AG. Furthermore, the recommendations were evaluated and compared offline. For the offline evaluation, a small percentage of customers' reviews of purchased products were removed beforehand. The resulting new rating dataset was then used as a test dataset and as an input for collaborative filtering. The results showed that the derived ratings are reliable and suitable for collaborative filtering. The offline evaluation showed that customers received recommendations for products they had already purchased, where their ratings had been previously removed from the rating dataset. This implies that the recommendations are relevant to the customers. It was also shown that item-based collaborative filtering performs better than user-based as an item is more frequently rated than a customer rates an item and therefore the item rating vector is less sparse in general.

Daniel Figia

Thema der Arbeit

Recommendation Engine: Kollaboratives Filtern basierend auf dem Kundenverhalten auf einer E-Commerce-Website

Stichworte

Recommendation Engine, Kollaboratives Filtern, Kundenverhalten, E-Commerce

Kurzzusammenfassung

Diese Arbeit hat untersucht, wie Bewertungen aus Clickstream-Daten abgeleitet werden können, um damit gute Empfehlungen zu generieren, indem diese als Eingabe für kollaboratives Filtern dienen. Immer mehr Menschen kaufen ihre Produkte online auf einer E-Commerce-Plattform. Deshalb werden die Produkt-Empfehlungen auf diesen Plattformen immer entscheidender für den Erfolg eines Unternehmens, denn desto besser die Empfehlungen, desto höher sind die Verkaufschancen. Um Empfehlungen zu generieren kann kollaborativem Filtern verwendet werden. Es versucht für jeden Kunden/Produkt ähnliche Kunden/Produkte zu finden basieren auf den Kundenbewertungen. Die Bewertungen können direkt durch ein Bewertungssystem gegeben werden oder indirekt abgeleitet werden, beispielsweise von Clickstream-Daten. Danach dienen die Bewertungen als Eingabe für kundenbasiertes und produktbasiertes kollaboratives Filtern, welches Empfehlungen generiert. Diese Arbeit hat das demonstriert mit Clickstream-Daten von novomind AG. Außerdem wurden die Empfehlungen offline evaluiert und verglichen. Dafür wurde vorher ein kleiner Prozentsatz der Bewertungen von Kunden zu ihren gekauften Produkte entfernt. Der daraus resultierende neue Bewertungsdatensatz wurde dann als Test-Datensatz für das kollaboratives Filtern verwendet. Die Ergebnisse zeigten, dass die extrahierten Bewertungen aus den Clickstream-Daten verlässlich sind und passend für das kollaborative Filtern. Die Offline-Evaluation zeigte, dass die Kunden Empfehlungen erhielten für Produkte, die sie bereits gekauft hatten, wo jedoch die Bewertungen für diese Produkte vorher aus dem Bewertungsdatensatz entfernt wurden. Daraus lässt sich ableiten, dass relevante Empfehlungen generiert werden können. Außerdem wurde gezeigt, dass produktbasiert kollaboratives Filtern besser ist als kundenbasiertes, weil ein Produkt öfter bewertet wird als ein Kunde ein Produkt bewertet. Damit hat der Bewertungsvektor eines Produktes generell eine größere Informationsdichte.

Contents

List of Figures

List of Tables

1 Introduction

In the time of the Internet, a large part of purchases is made online. More and more people want to buy articles online comfortably from their phones or computers. 18.8 % of all retail sales were made online as the graph in Figure 1.1 shows. The trend has been increasing steadily since 2015 and it will presumably continue to increase.

Online shopping provides a lot of advantages such as ordering from multiple places, a simpler way of searching for articles and direct checkout. Thus, companies strengthen their online presence and try to improve the user experience to increase their sales. There is almost no big company that does not have a webshop.

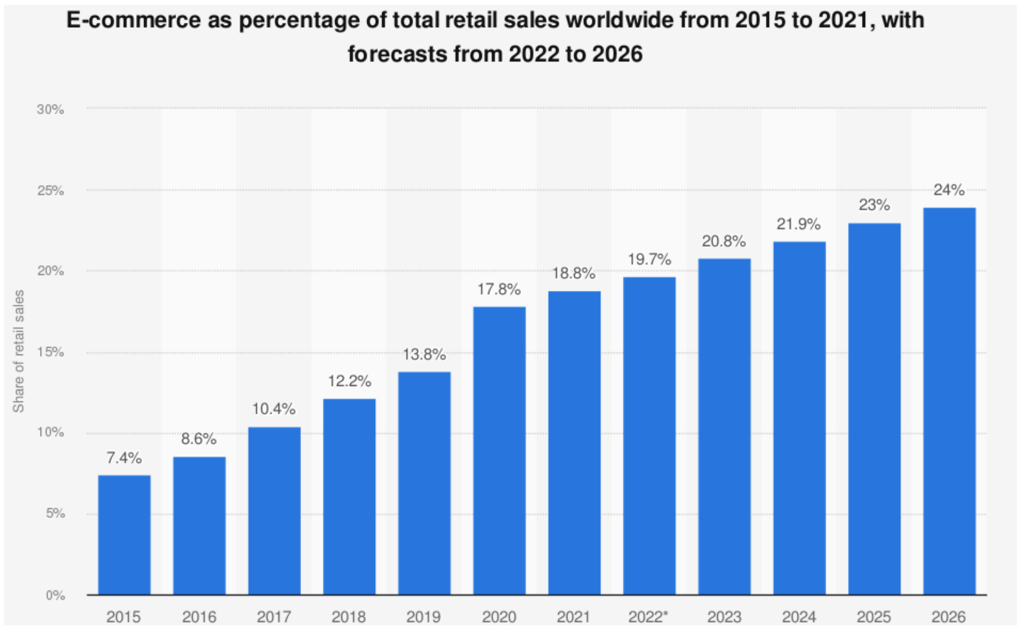


Figure 1.1: E-commerce as percentage of total retail sales worldwide from 2015 to 2021, with forecasts from 2022 to 2026 [eMa22]

A primary role of the user experience is the recommendations. Good recommendations will satisfy the users as they get to know new potentially interesting products and/or further fitting products to their latest transactions. Recommendations in the offline world are mostly made by statistical indicators such as top sellers or by a seller who advises a customer individually. Statistic indicators may address a large crowd but they do not address individuals. A single seller who advises a customer individually is expensive and not effective because the seller cannot analyse and advise the customers all at once. However, these problems can be better solved in the online world.

Most web shops provide a rating system where the customers can give their opinion about an item through a rating scale. Ratings made directly through a rating system are called explicit ratings. If enough users use the rating system, then these ratings can be used to generate user-specific recommendations. However, a rating system is not always available or the ratings the users made are sparse. In that case, other data, such as tracking data, clickstream data or purchase history, can be used to create ratings. A rating for a user and an item can be based on the number of views or on the purchase status (purchased or not purchased). This kind of rating is called implicit rating.

The ratings are usually made by a recommendation system that uses one or more techniques from content-based filtering, collaborative filtering or a combination of both. Content-based filtering finds similarities between products by using their properties and then recommends the products with the highest similarity (*"products like this"*). Collaborative filtering finds similarities between users by using their ratings and then recommends products that similar users rated positively (*"products other users also liked"*).

1.1 Problem statement

The company *novomind AG*[1] provides different kinds of e-commerce solutions among others also webshops. Well-known companies such as the drugstore chain *Mueller*[2] or the outdoor equipment retailer *Globetrotter*[3] have a webshop provided by novomind.

Webshops from novomind usually have a rating system. But these rating systems are not used by all users and therefore ratings are sparse. Most users could not receive recommendations because there are no ratings available from them and most items would

[1]www.novomind.com
[2]www.mueller.de
[3]www.globetrotter.de

not be included in the recommendation process due to missing ratings. Despite that, a webshop from novomind collects information about almost every click a user makes. For example: When a user used the search, used the filter, viewed a product, submitted an order or logged in/out, then specific information about the activity is collected and sent into a Hadoop cluster. This data is called clickstream data. The data can be used to extract implicit ratings by analysing the behavior of the users or analysing the products the user viewed, added to the basket or purchased.

Currently novomind only uses content-based filtering based as a recommendation technique. A user receives recommendations which are similar to the products the user purchased. The problem is that the recommendations are restrictive and they do not go above the interest of the users. Collaborative filtering solves this problem as it does not compare the properties of the products to generate recommendations, but it relies on the ratings of the users. It groups similar users together based on their ratings and then recommends products which are popular within these groups.

Furthermore, content-based filtering requires knowledge about the domain, because the properties of the products must be analyzed. Selecting unsuitable properties could lead to bad recommendations, thus they have to be selected carefully. If a shop wants to include recommendations based on content-based filtering, then it would be more expensive and time-consuming. In contrast, collaborative filtering can be used directly for all shops, once it is implemented.

In conclusion, there are three problems:

- The currently used recommendation technique (content-based)
  - is restrictive,
  - and cannot be used effectively without domain knowledge and prior analysis.
- There are not enough explicit ratings available.

1.2 Goal and methodolgy

The goal of this thesis is to find an approach to solve these problems with collaborative filtering techniques and implicit ratings. The implicit ratings will be extracted from clickstream data by finding behavior patterns which are used to derive ratings. Firstly, the clickstream data from novomind is analyzed and preprocessed. Then two algorithms, which can create ratings from clickstream data, will be analyzed, adjusted, implemented and evaluated.

The hypothesis examined in this thesis is **whether collaborative filtering can be used effectively with the clickstream data provided from novomind**. The effectiveness will be measured with different metrics such as recall, precision or intra-diversity. All used metrics will be explained in detail in the sections *Evaluation of collaborative filtering* 2.4 and *Evaluation methods* 6.

1.3 Thesis outline

The second chapter *Theoretical Foundation* 2 provides fundamental information about big data (clickstream data), Hadoop (location of the clickstream data), classification and recommendation systems (especially collaborative filtering) and the evaluation of collaborative filtering techniques.

The third chapter *Related Work* 3 describes the algorithm used for the extraction of the ratings. The algorithms will be described in detail and demonstrated by an example.

The fourth chapter *Data* 4 gives more insight into the company novomind and describes their invention *Multi-Channel-Selector* 4.1.1 and their clickstream data structure. The data which is used for the evaluation is also described.

The fifth chapter *Data transformation* 5 shows how the clickstream data can be transformed so that it can be used effectively. Furthermore, it shows how to clean faulty data.

The sixth chapter *Evaluation methods* 6 lists and explains the methods for the evaluation.

The seventh chapter *Implentations* 7 is all about the implementation of the two algorithms. Different modules were implemented and they will be elaborated in this chapter.

In the eighth chapter *Evaluation* 8 the results of the evaluation are shown and discussed.

The final ninth chapter *Conclusion* summarizes the results and discusses problems and future work.

2 Theoretical Foundation

2.1 Big Data

This thesis will use clickstream data to create implicit ratings. Clickstream can be categorized as big data. This section explains what big data is and why clickstream data belongs to big data.

Data plays a big role nowadays. It is collected in various forms for almost every reason: health, e-commerce, finances, climate, sport, social networks and so on. According to a statistic from Statista [IS21] (see Figure 2.1), the volume of data globally was approximately 2 zettabytes in 2010. In the year 2017, the volume multiplied by 13 and increased to 26 zettabytes. The volume of data collected for 2025 is estimated to be 181 zettabytes.

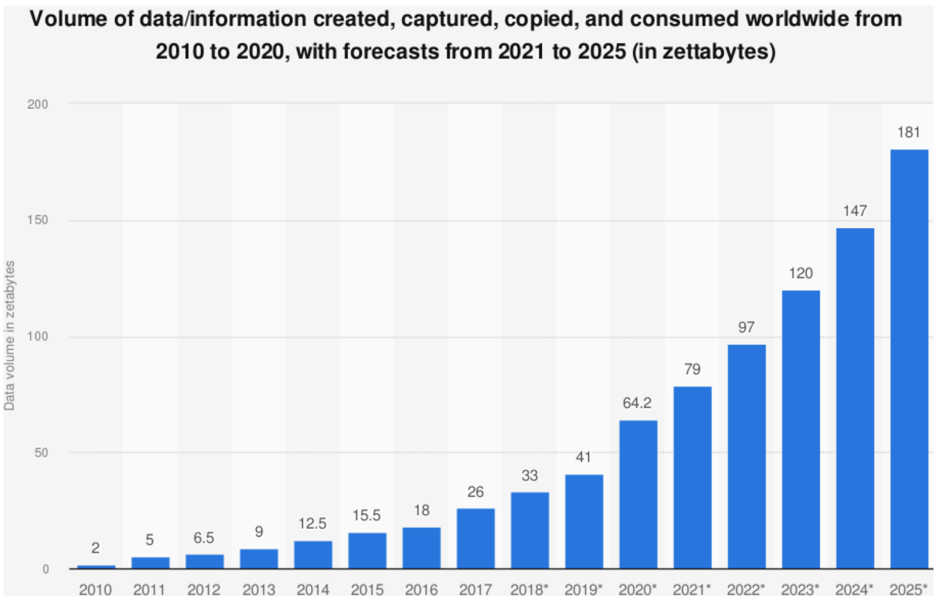


Figure 2.1: Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025 [IS21]

It is a challenge to manage the increasing volume of data and thus a new research field has been introduced: **Big Data** [FM16]. Big Data is a relatively new research field that investigates how a large volume of various data from different sources can be managed. Big Data can be characterised with five **V´s**:

Volume: The data volume is very large. The size of the data is usually between terabytes and zettabytes.

Varity: The data is various. It can be structured (e.g. relational tables), semi-structured (e.g. XML, JSON) or unstructured (e.g. multimedia like videos, images, text)

Velocity: The data must be read and analyzed as fast as possible (in extreme cases in real-time)

Value: The data and the finished analyses should bring value to the company or organisation.

Veracity: Some data is often inaccurate or incorrect. Therefore the significance of the results of the data analyses must be evaluated with specific algorithms.

2.1.1 Clickstream data

If a user clicks through a website, then every click can be tracked and certain information about the current activity can be collected. The data, which is gathered this way, is called **clickstream data**. A mechanism in the background is triggered if a user clicks somewhere and sends information about the current activity of the user to a database. This could be for example: the current URL, timestamp, products in the basket, device details or/and browser details. If enough data is collected, then different analyses can be done to extract information about the users such as navigational behavior patterns, most frequently visited sites or common processes.

Clickstream data is big data because its characteristics fit the five V´s: The **volume** is very large and it grows larger over time. It is **various** as the data can be structured in different ways and also can also contain multimedia (e.g. upload of images or videos). In some cases, the data must be analyzed as fast as possible (e.g. item recommendations) (**velocity**). The result of the analysis brings **value** to the company by improving the experience for their users. The data is also often inaccurate because the mechanism fails to recognize a trigger or the user uses browser plugins such as ad blocker or cookie blocker, which can cause problems (**veracity**).

2.2 Hadoop

The clickstream data of novomind is persisted in a Hadoop [Fou10] cluster. The data can be accessed with Impala, a fully integrated and parallelized SQL database engine for Hadoop. This section explains what Hadoop is and briefly describes how Hadoop works.

Hadoop [Fou10] is a framework which provides tools to persist and process big data. The Hadoop framework consists of multiple core components. Two of the most important core components are the distributed file system **Hadoop Distributed File System** and an implementation of the **MapReduce** algorithm. Furthermore, there are multiple open-source tools to access and analyse the data, which use Hadoop as a platform and are part of the Hadoop ecosystem [FP18].

Hadoop uses commodity hardware (hardware that is inexpensive, widely available and can be easily replaced with other hardware of its own type) and enables parallel processing. It usually runs on a horizontally scalable cluster and can be extended or downsized by changing the number of nodes. So instead of buying new and better hardware (scaling up), Hadoop extends its cluster with more of the same type of hardware (scaling out) [FP18].

The **HDFS** (Hadoop Distributed File System) is a file system, which is distributed over the cluster. It consists of one master node and one or more data nodes. The master node manages all metadata of the file system, while the data nodes manage the memory space that was allocated to them. The data nodes are replicated to improve reliability.

2.2.1 MapReduce

MapReduce is a programming model that processes a large volume of unstructured or semi-structured data. MapReduce uses the distributed file system and distributes the calculation to multiple systems for parallel execution. As the name already implies, MapReduce consists of three operations (map, shuffle and reduce). The map function receives input data and creates key-value pairs. The shuffle function groups the values which have the same keys together. The result of the map function serves as the input for the reduce function which reduces each value group to a single value. An example is shown in Figure 2.2. This example aims to group the letters to their frequency. A string of three lines serves as input. This file is preprocessed and prepared for mapping by splitting the lines. On each line, the frequency of each letter will be counted and the frequency will be mapped to the letter. The shuffle function groups all letters together and puts their frequencies into a list. Finally, the reducing function adds up the frequencies of each key.

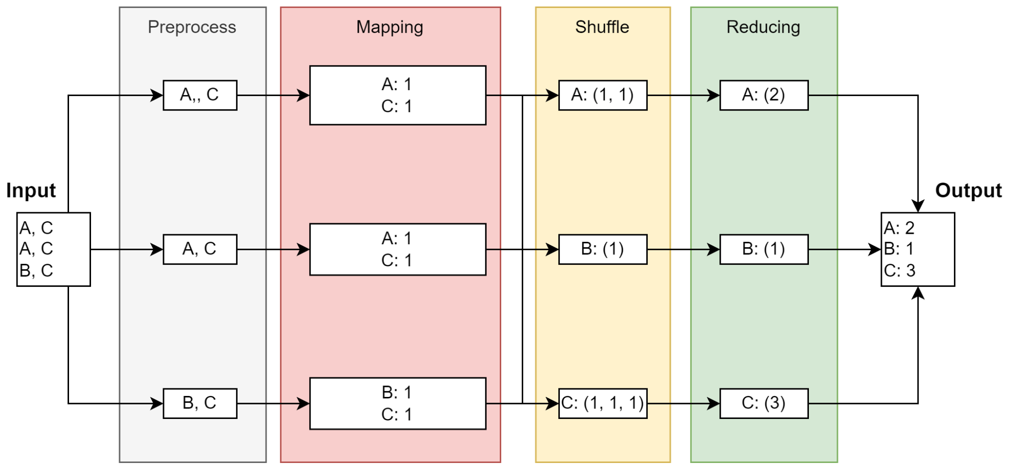


Figure 2.2: MapReduce example [IFO15]

2.2.2 Impala

Impala [Fou] is an open-source, fully integrated and parallelized SQL database engine. It allows the user to read and write data from the HDFS. Impala uses the advantages which Hadoop has to offer and therefore it belongs to the Hadoop ecosystem. The engine provides typical features of an analytical database and it is optimized for performance in a multi-user system. Two major advantages of Impala are its high read performance and the familiar SQL syntax. However, the two major disadvantages of Impala are that it does not support indexing and it cannot update or delete any data.

2.3 Recommendation System

The implicit ratings created from the clickstream data of novomind will be used to create recommendations. This will be done by memory-based collaborative filtering which is a technique from recommendation systems. This section describes and compares the techniques a recommendation system uses to create recommendations with a focus on the memory-based collaborative filtering technique.

A recommendation system [IFO15] generates personalized item recommendations by predicting the interest of a user for each available item. The recommendations are not just based on statistics such as top-seller or most-viewed but also on the implicit and explicit item ratings. Explicit ratings are ratings that were made by the user directly through a rating system (e.g. stars, likes/dislikes or comments) and implicit ratings are ratings which only can be derived from user behavior (e.g. clickstream data or purchase history). Therefore the primary difference between the both of them is: explicit ratings reflect the true opinion of the user, while implicit ratings assume the opinion of the user.

Recommendation systems are becoming more popular in e-commerce as they are able to recommend more user-specific items, thus increasing the probability of sales. A recommendation system uses one or more filtering techniques to generate recommendations. There are three main techniques: *Content-based filtering*, *Collaborative filtering* and *Hybrid filtering*. These techniques are described in the following sections. An overview of these techniques is shown in Figure 2.3.

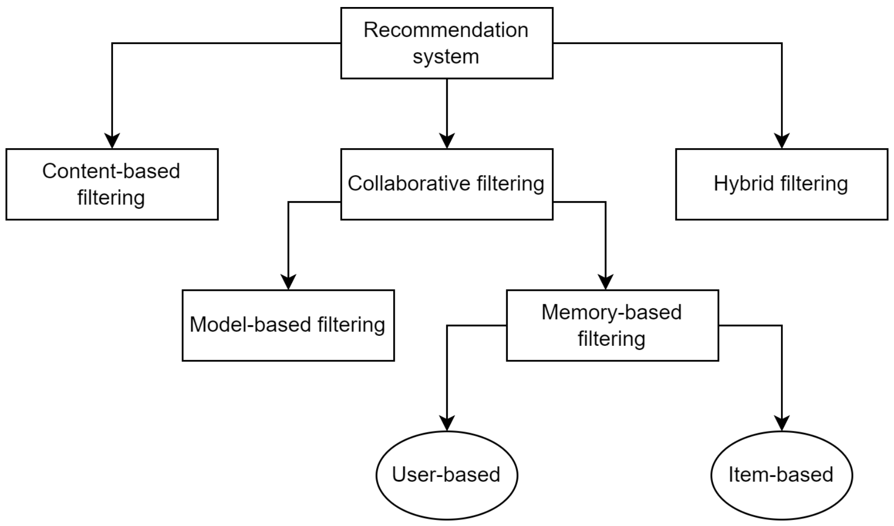


Figure 2.3: Recommendation system: Filtering techniques [IFO15]

2.3.1 Content-based filtering

The content-based filtering technique focuses on the properties of items in order to generate predictions. Therefore, content-based filtering is **domain-driven** and in order to implement this technique, the properties of the items need to be analyzed beforehand. Items that a user has rated positively, will be used to generate new item recommendations. The recommendations are generated by comparing the properties of all positively rated items with the properties of the items the user has not rated yet.

Here is an example with implicit ratings: Imagine a clothing store. A user mostly buys sports clothing for women in size M. The following categories can be derived from this: gender (female), size (M) and category (sport). A content-based filtering approach would take these three properties and search for items with similar properties and the user would receive recommendations for items with similar properties. Another example with explicit ratings: A YouTube[1] user mostly likes videos with animals and healthcare. A recommendation system with content-based filtering could now recommend more videos of animals and healthcare to this user.

The **advantages** of content-based filtering are that it includes all items, even items with missing ratings, and it can react in a short time span to new ratings. The user also does not have to share profile information as the technique only focuses on the items which the user has interacted with.

The **disadvantages** of content-based filtering are that it requires knowledge of the domain and the domain has to be analyzed in advance. This is expensive, time-consuming and it has to be set up for each new e-commerce website due to the difference of item properties, despite the website being in the same domain. Furthermore, there is another problem called over-specialization: The recommendations to a user will tend to be very similar to each other, which decreases the diversity. Items that could also be interesting for the user, but are not similar to the items the user rated positively, will hardly be recommended.

2.3.2 Collaborative filtering

Collaborative filtering is in contrast to content-based filtering **domain-independent**. The goal of collaborative filtering is to create recommendations only based on the user's

[1] www.youtube.com

ratings. The ratings are usually represented as a matrix, where the axes are the IDs of the user and items. A value in the matrix describes the rating of a user and an item. This matrix is called the **user-item matrix**. The recommendations are made either by prediction or by creating a top-n list. The collaborative filtering techniques can be further split into two more techniques: *Memory-based* and *Model-based*.

Memory-based filtering

Memory-based filtering techniques use historical user data to compute a similarity matrix between users or items. Then, a neighbourhood for each user or item is formed by grouping the users/items with the highest similarities together. Finally, recommendations can be made based on these neighbourhoods. There are two general approaches for memory-based filtering: user-based and item-based [Her+04].

User-based collaborative filtering aims to find similar users derived from a user-item matrix. A neighbourhood is formed for each user. The ratings within one neighbourhood are then combined and used to predict new preferences for each user.

The similarity between two users is calculated with similarity functions such as *Pearson correlation coefficient* [FPP07], *constrained Pearson correlation* [SM95], *Spearman rank correlation* [Doe08] or the *cosine similarity* (equation 2.1).

$$Cosine\ similarity(a,b) = \frac{\sum\limits_{i \in I} a_i * b_i}{\sqrt{\sum\limits_{i \in I} {a_i}^2} * \sqrt{\sum\limits_{i \in I} {b_i}^2}} \tag{2.1}$$

where *(a, b)* $\in$ *Users* and *I* = *Item preferences*

Afterwards, a neighbourhood is formed by finding the n-most similar users for each user. Once a neighbourhood is built, the preferences for other items can be estimated with the formula 2.2 below.

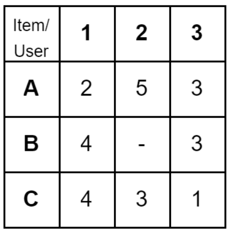
$$P_{ij} = \bar{P}_i + \frac{\sum\limits_{n \epsilon N_i} Similarity(i,n) * (P_{nj} - \bar{P}_n)}{\sum\limits_{n \epsilon N_i} Similarity(i,n)} \tag{2.2}$$

The formula calculates a preference P for user i and item j by calculating the weighted arithmetic mean of all ratings from every neighbour n from the neighbourhood N of user i. After the preference for each item is calculated, k items with the highest preferences are chosen for the final recommendations.

Item-based collaborative filtering is very similar to user-based, but for each item, a set of similar items is formed instead of forming a neighbourhood of users for each user. The similarity functions are the same as for user-based, but the most popular ones for item-based are *Pearson correlation coefficient* [FPP07] and *cosine similarity* (equation 2.1). The preferences for each user and item can be calculated with the formula 2.3 below. The variable s stands for an item from the similar item set S for item j.

$$P_{ij} = \frac{\sum\limits_{s \epsilon S_j} Similarity(j,s) * P_{is}}{\sum\limits_{s \epsilon S_j} Similarity(j,s)} \tag{2.3}$$

Example: The user-item matrix shown in 2.4 consists of three items (1, 2, 3) and three users (A, B, C). One rating for user B and item 2 is missing. The missing rating should be calculated with user-based and item-based collaborative filtering whereas the cosine similarity should be used as a similarity function.



Item/ User	**1**	**2**	**3**
A	2	5	3
B	4	-	3
C	4	3	1

Figure 2.4: User-item matrix example

With **user-based** collaborative filtering the missing rating is estimated by first calculating the similarities between user B and the other users. The missing rating *B2* is set to zero:

Item/ User	1	2	3	*Avg*
A	2	5	3	*3.33*
B	4	0	3	*3.5*
C	4	3	1	*2.66*

Figure 2.5: User-item matrix user-based example

$$Similarity(B,A) = Cosine\ similarity(B,A) = \frac{4*2+5*0+3*3}{(\sqrt{4^2+0^2+3^2})*(\sqrt{2^2+5^2+3^2})} \approx 0.55$$

$$Similarity(B,C) = Cosine\ similarity(B,C) = \frac{4*4+0*3+3*1}{(\sqrt{4^2+0^2+3^2})*(\sqrt{4^2+3^2+1^2})} \approx 0.75$$

A neighbourhood for user B is then formed with the n most similar users. Due to the small number of users in the matrix, all other users can be included in the neighbourhood. In the end, the similarities and the ratings of the other users are used to estimate the preference for user B and item 2 with the formula 2.2:

$$P_{B2} = 3.5 + \frac{0.55*(5-3.33)+0.75*(3-2.66)}{0.55+0.75} \approx \underline{\underline{4.4}}$$

With **item-based** collaborative filtering the missing rating is estimated by first calculating the similarities between item 2 and the other items:

Item/ User	1	2	3
A	2	5	3
B	4	0	3
C	4	3	1

Figure 2.6: User-item matrix item-based example

$$Similarity(2,1) = Cosine\ similarity(2,1) = \frac{5*2+0*4+3*4}{(\sqrt{5^2+0^2+3^2})*(\sqrt{2^2+4^2+4^2})} \approx 0.63$$

$$Similarity(2,3) = Cosine\ similarity(2,3) = \frac{5*3+0*3+3*1}{(\sqrt{5^2+0^2+3^2})*(\sqrt{3^2+3^2+1^2})} \approx 0.71$$

The next step is to form an n-most similar item set for item 2. The number of items is small and therefore all other items are included in the similar item set. The last step is to estimate the preference for user B and item 2 with the formula from 2.3:

$$P_{B2} = \frac{0.63*4+0.71*3}{0.63+0.71} \approx \underline{\underline{3.47}}$$

It is not clear which technique is better as it depends on the data set and the recommendation system. Many papers have different results when it comes to comparing user-based and item-based collaborative filtering. For instance, in the paper [GK19] the authors show that user-based has lower errors than item-based. Paper [XE18] displays different results where both techniques provided good results.

When a system has more users than items, then item-based is faster than user-based, since lesser iterations are required and certainly user-based is faster if it is the other way around. Furthermore, item-based could be superior to user-based, if an item is more often rated, than a user rates an item. In this case, the rating space of the items is larger and provides more information which might lead to better recommendations.

The problem with memory-based filtering is that a large memory volume is required for storing the user-item matrix. Moreover, the larger the user-item matrix is, the longer it takes to compute.

Model-based filtering

Model-based filtering techniques create a model based on the ratings of the users. The model-building process is usually done by machine learning or data mining techniques such as decision trees, clustering, artificial neural networks or association rules. Model-based filtering techniques aim to solve the major drawback of memory-based filtering techniques which is the large size of required memory. The pre-computed models, which use lesser memory, make the recommendation process faster and memory-friendly.

Advantages of collaborative filtering

Domain-independence: One of the major advantages of collaborative filtering is that it is domain-independent. Once collaborative filtering is implemented, it can be used for every e-commerce system and there is no need for a domain expert. The only thing that has to be pre-processed is the rating data, which can vary from system to system.

Diversity: The item recommendations are more diversified. A user receives recommendations that may or may not suit their preference. This offers users the chance to explore what they could be interested in. For example: In an online bookstore, a user usually buys crime genre books. With content-based filtering, the user would receive further crime genre book recommendations, but with collaborative filtering, there is a chance that the user may receive a recommendation for a book from another genre. That is because the collaborative filtering approach involves other users with similar ratings and does not only focus on one user and their ratings.

Disadvantages of collaborative filtering

Missing Data: If there are not enough ratings, then collaborative filtering will fail, because the user-item matrix is too sparse. This problem is also called **Cold-start**.

Scalability: The computation time will increase with the number of users and items. This is especially crucial in memory-based filtering as the neighbourhood must be updated with every new rating and this process is time-consuming. This problem can be reduced by recomputing the user-item matrix in batches instead of updating it when a new rating is made.

Unable to differentiate similar items: The properties of an item are unknown in collaborative filtering. Therefore items cannot be differentiated. And that can lead to a problem as a user receives an item recommendation which is very similar to an item the user already bought.

Fake reviews: If only the highest-rated items are recommended, then low-rated items are neglected. This could be fair because usually, an item is low-rated for a reason. However, nowadays some sellers buy fake reviews [HHP21] to promote their products, regardless of whether the products are good or bad. These fake reviews cause problems and one of the problems is, they distort the recommendations made by collaborative filtering.

2.3.3 Hybrid filtering

Hybrid filtering techniques aim to combine the best of content-based and collaborative filtering techniques. The goal is to optimize the recommendation systems in computation time and outcome. However, every technique has disadvantages and limits. If the techniques are combined, some of these problems can be reduced. To give an example: The Cold-start problem from collaborative filtering can be solved by using content-based filtering at the launch of a new system, where no rating data is available. After enough rating data is collected, then collaborative filtering techniques can then be included in the recommendation process to make the recommendations more diverse.

2.4 Evaluation of collaborative filtering

This thesis will evaluate the results of collaborative filtering. This section describes how the results of collaborative filtering can be evaluated.

The evaluation of collaborative filtering can be done offline and online. The offline evaluation is useful to compare two or more recommendation systems, but it cannot evaluate the user´s satisfaction with the recommendations. This can only be done with real users and therefore, an online evaluation is necessary [Jal+18] [Her+04].

2.4.1 Offline evaluation

The common offline evaluation is usually similar to machine learning evaluation techniques, where the data is split into a training and a test set. The recommendation system is trained with a training set and then evaluated with a test set. The performance of a recommendation system can be measured with the predicted ratings generated by the test set [Her+04]. The metrics can be grouped into five groups: accuracy, ranked-based, diversity, novelty and coverage [Jal+18]. The accuracy metrics are the relevant ones for this thesis, especially the metrics *recall* and *precision*, hence these metrics will be explained more thoroughly in the following section 2.4.1.

Prediction accuracy: Measures how accurate the predicted ratings are, by comparing predicted ratings with real ratings.

Ranking accuracy: Measures the accuracy between a ranked list of predicted ratings and a ranked list of real ratings. A ranked list is a list where the order of items is relevant.

Diversity: Measures how different the items recommended to a user are. The similarity between each item can be calculated by comparing the item properties.

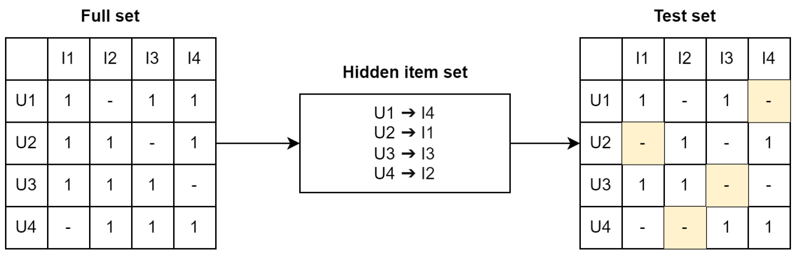
Coverage: Measures the variety and scope of the recommended items. Coverage metrics determine how many of the total items are included in the recommendation process.

Novelty: Measures how unusual or new the recommendations for a user are. A novel recommendation is a recommendation for an item that the user is not aware of before.

Accuracy metrics

If a full set of real data is available, then accuracy metrics are most accurate. For recommendation systems, a set of real data would be explicit data, which consists of actual user ratings. Unfortunately, explicit data is not always available or to sparse, because users do not always rate the items they like. However, implicit ratings such as historical purchase data or clickstream data are often available in large quantities. The problem with implicit data is that it does not give information about whether a user likes the product or not. If a user purchases a product, then it is possible that the user is not satisfied with the product in the end or the user purchased the product just as a gift. Nonetheless, if the product is viewed and purchased by a user, then the probability that the user is interested in the product is high. If the assumption is made that a purchased item is liked by the user, then the metrics *recall*, *precision* and *F1-Score* can be used. They originate from statistical decision theory [Ber18] and are widely used in the information retrieval field [Sal92] [VM03]. They are also commonly used to evaluate the top-N recommendations.

The first step to calculate recall and precision is to identify relevant items for each user. Then a **hidden item set** is built by selecting a subset of these relevant items. The ratings for each user and relevant item from the hidden item set are then removed from the user-item matrix. The resulting user-item matrix forms the **test set**, which is used to train the collaborative filtering algorithm. Figure 2.7 shows a simple illustration of the procedure:



Full set

	I1	I2	I3	I4
U1	1	-	1	1
U2	1	1	-	1
U3	1	1	1	-
U4	-	1	1	1

Hidden item set

U1 ➔ I4
U2 ➔ I1
U3 ➔ I3
U4 ➔ I2

Test set

	I1	I2	I3	I4
U1	1	-	1	-
U2	-	1	-	1
U3	1	1	-	-
U4	-	-	1	1

Figure 2.7: Example of splitting a data set into a test set and a validation set

The output of the collaborative filtering should be a top-N recommendation list. Finally, recall and precision can be calculated with the following formulas:

$$Hits = \sum_{i \in A} |H_i \cap Top\_N_i| \tag{2.4}$$

$$Recall = \frac{hits}{\sum_{i \in A} |H_i|} \tag{2.5}$$

$$Precision = \frac{hits}{\sum_{i \in A} |Top\_N_i|} \tag{2.6}$$

where:
A = all users with at least one hidden item
H_i = the hidden items of user i
Top_N_i = Top-N list for user i

Hits describes how many of the hidden items are recommended again to the corresponding user *i*. *Recall* is the proportion of hits to the total number of hidden items. *Precision* is the proportion of hits to the total number of recommended items. A high recall indicates that the recommendation system can find relevant items for the users. High precision means that the recommendation system does not need a high N (number of recommendations) to find relevant items for the users.

Recall and precision are dependent on each other. If the recall increases, the precision will most likely decrease. A high number of recommendations makes it more probable to find a relevant item (increases the recall). Still, the precision will decrease if there are more irrelevant items in the top-N recommendation list. Therefore, a third metric combines *recall* and *precision* in order to receive a unique value. This metric is called *F1-score*. The formula is shown below in equation 2.7 [VM03].

$$F1 - score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{2.7}$$

Example: The hidden items of a user are [A, B, C] (H$_i$) and a list of recommendations for the same user are [A, C, F, E] (R$_i$). Then *Hits* would be *2* because A and C are present in both lists. *Recall* would be **0.67**, because the size of the hidden items is 3.

Precision would be **0.5** as the size of recommendations is 4. Finally, the F1-score would be *(2 * 0.67 * 0.5) / (0.67 + 0.5) =* ***0.57***.

2.4.2 Online evaluation

The actual value of the generated recommendation of a recommendation system can only be determined with an online evaluation, which means that real users and a live system are required to observe the behavior of the users. Users could give explicit feedback to the recommendations or implicit feedback could be determined by tracking the click-through rate. A very commonly used method is the A/B test. In an A/B test, the users are split into two groups. One group will receive recommendations as before and the other group receives recommendations from the new system. After a specific period, the groups can be compared and ultimately conclusions could be drawn [Jal+18] [Her+04].

The problem with online evaluation is, it is expensive and time-consuming. A client with a real live system is necessary and the new recommendation system must be well-tested because an error could be very costly. In the worst case, the client loses customers and this must be prevented by all means. Another problem is that it takes time until the results can be analyzed.

2.5 Classification

A major topic of this thesis is the extraction of implicit ratings from clickstream data. One algorithm **KimRec**, which will be elaborated later in detail in sections 3.1 and 7.5, uses classification techniques to predict the probability of a basket placement based on the navigational and behavioral pattern of the users. This section gives an overview of the classification techniques used in this thesis.

Classification [Alp10] is a type of supervised machine learning. A model is trained with labelled data (each data record has a class). This model is then used to predict the class of unlabelled data. There are three different types of classification:

Binary classifcation Prediction of the class for a data record, whereby the class is binary (e.g. mail → spam/no spam, patient record → diseased/healthy).

Multiclass classification Prediction of the class for a data record. There are several classes to choose from instead of only one or two classes (e.g. a picture of an animal → cat/dog/fox/badger).

Multilabel classification Prediction of multiple classes for a data record (e.g. the tags for an article or the categories of a game).

Often a model predicts a probability for each class and the highest probability decides the final class. The classification techniques used in this thesis are:

Logistic Regression Logistic Regression is a simple and effective method to model the probability of a binary outcome. The model aims to find the best-fitting model which describes the relation between features and classes. It is built by a set of weights or coefficients, which multiplies the feature values and adds them up. The final value is mapped to a value between 0 and 1, representing the predicted probability through a function called the *logistic function*. While developing or training a classification model, logistic regression models are often used as the baseline for performance evaluation [HL13].

Decision Tree A decision tree is a tree graph where every node represents a decision. The graph has a root node, which is the starting point. The nodes without children are the classes. If traversing through the tree, then in each node a decision must be made to continue. If a node does not have any children, the node determines the final class. These decisions are based on a particular feature value. Figure 2.8 is an example of a decision tree, which determines an animal based on its physical characteristics. [Bre84].

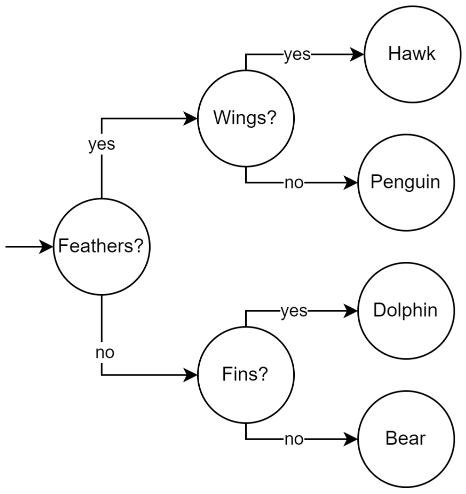


Figure 2.8: Example of splitting decision tree.

Random Forest A random forest is an ensemble of decision trees, where each decision tree is trained on a different subset of data. Instead of relying on one decision tree prediction, it relies on many decision trees' predictions and combines them into a final one. [Bre01].

Adaptive Boosting Adaptive Boosting (AdaBoost) uses an ensemble of weak models. A weak model is a model that is just slightly better than random guessing (e.g. simple linear regression model [HTF09]). AdaBoost uses the models iteratively, each model observes the mistakes of the other models and uses the observation to improve their model. A weight is assigned to each model and the more accurate the model is, the higher the weight [FS97].

Gradient Boosting Gradient Boosting also uses an ensemble of weak models. First, it selects a base model (e.g. a decision tree). This model will be the first member of the ensemble. Then it starts to train one model on a set of training data. In the next step, it uses the trained model to make predictions and calculates the losses between the predicted values and the true values. Afterwards, it trains a new model using the losses of the previous model as the target variable. The goal is that the new model reduces the loss. The new model is then added to the ensemble. This procedure is repeated until the ensemble reaches a specific size. Each member of the ensemble has a weight and the final prediction is made by combining all predictions [Fri01].

Multilayer Perception A multilayer perception (MLP) is a kind of artificial neural network and it is composed of perceptrons. The data in an MPL flows only through one direction and that means it is a feedforward neural network. It is also trained for a specific task such as classification. The weights of the connections of the neurons within the neural network are adjusted to the input data and the desired outcome [RM86].

3 Related work

3.1 Development of a recommender system based on navigational and behavioral patterns of customers in e-commerce sites

The paper of the *Korea Advanced Institute of Science and Technology* [Kim+05] proposed a recommendation system based on the navigational and behavioral patterns of the user from an e-commerce site. The authors did not name their system, therefore it is referred to as **KimRec** in this thesis. Their approach was to extract navigational and behavioral patterns from clickstream data and then use this pattern to create a user-item matrix, which serves as the input for collaborative filtering. The extracted patterns are:

Click type: Binary variable to indicate, if the user got the item via browsing (0) or searching (1).

Number of visits: Total item visits of the user.

Length of reading time: Total item viewing time.

Category click ratio: Each item has a category and each category can have a subcategory. So the ratio of how often a customer visits a category can be calculated for each visited item per customer.

Basket placement status: Binary variable to indicate whether the user added the item to the basket.

Purchase status: Binary variable to indicate whether the user purchased the item.

The general process consists of three phases:

1. First, the pattern data is extracted from click-stream data.

2. Then the rating for each user and every item is estimated by using the extracted pattern. The estimated ratings are wrapped in a user-item matrix.
3. Lastly, the user-item matrix serves as an input for user-based collaborative filtering, which will then return a top-N recommendation list.

The second phase is the most interesting one. Here the authors used formulas and machine learning to estimate the ratings of a user for every item, which was clicked by the user. If the user clicked on an item, then there are three cases: The user has purchased the item, the user has added the item to the basket or the user has only viewed the item without adding it to the basket or purchasing it.

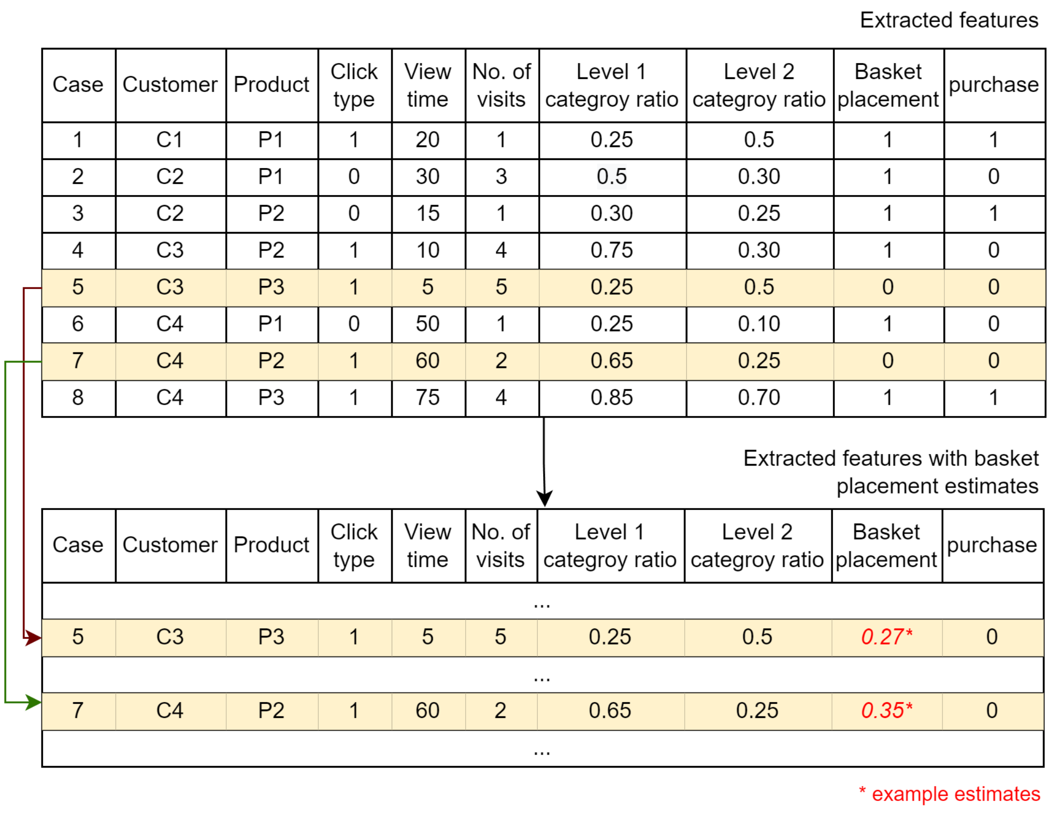
If a user purchases an item then it is the best indicator that the item is relevant for the user. Therefore the ratings for the first case are set to 1, which is the highest rating. For the second case the rating p for item i is calculated by dividing the total number of cases in which item i is purchased ($Purchased\ count_i$) by the total number of cases in which item i is added in the basket ($Basket\ count_i$):

$$p_i = \frac{Purchased\ count_i}{Basket\ count_i} \tag{3.1}$$

The ratings for the third case cannot be easily estimated by using statistics. Therefore, a machine learning algorithm is used to estimate the ratings. The inputs for the machine learning algorithm are the extracted pattern data (excluding the purchased status and basket placement status), where the target variable is the probability of basket placement. The rating for an item which is clicked but not added to the basket ($\bar{p_i}$) is then calculated with the formula 3.2. The ratings for each user and item are then put into a user-item matrix.

$$\bar{p_i} = p_i * basket\ placement\ probability\ estimate \tag{3.2}$$

Example: The process is illustrated in Figures 3.1 and 3.2 . Figure 3.1 shows extracted features that are assembled in a table format. Cases 5 and 7 do not have a basket placement and thus the basket placement will be estimated. The user-item matrix shown in Figure 3.2 is created after the estimation of the basket placement. If a user purchased an item, then the rating will be 1 [(C1, P1) = 1, (C2, P2) = 1, (C4, P3) = 1]. If a user added an item to the basket but has not purchased it, the rating is calculated by using the equation from 3.1 [(C2, P1) = 1/3, (C3, P2) = 1/2, (C4, P1) = 1/3]. The ratings for the user C3 and item P3 and user C4 and P2 are calculated with the equation from 3.2 [(C3, P3) = 1 * 0.27 * 1, (C4, P2) = 0.50 * 0.35].


Extracted features

Case	Customer	Product	Click type	View time	No. of visits	Level 1 categroy ratio	Level 2 categroy ratio	Basket placement	purchase
1	C1	P1	1	20	1	0.25	0.5	1	1
2	C2	P1	0	30	3	0.5	0.30	1	0
3	C2	P2	0	15	1	0.30	0.25	1	1
4	C3	P2	1	10	4	0.75	0.30	1	0
5	C3	P3	1	5	5	0.25	0.5	0	0
6	C4	P1	0	50	1	0.25	0.10	1	0
7	C4	P2	1	60	2	0.65	0.25	0	0
8	C4	P3	1	75	4	0.85	0.70	1	1

Extracted features with basket placement estimates

Case	Customer	Product	Click type	View time	No. of visits	Level 1 categroy ratio	Level 2 categroy ratio	Basket placement	purchase
...									
5	C3	P3	1	5	5	0.25	0.5	*0.27**	0
...									
7	C4	P2	1	60	2	0.65	0.25	*0.35**	0
...									

* example estimates

Figure 3.1: Extracted features table

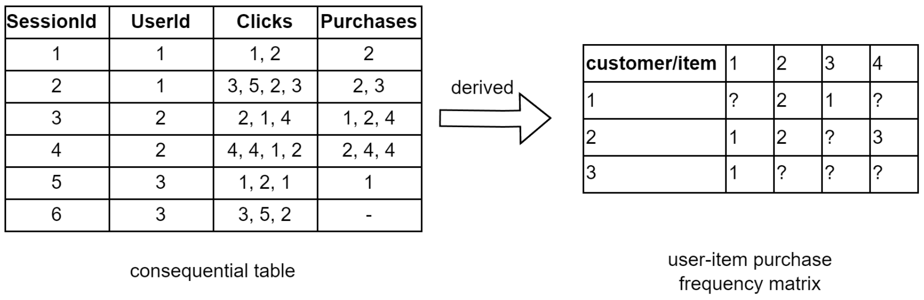
	P1	P2	P3
C1	1	0	0
C2	0.33	1	0
C3	0	0.50	0.27
C4	0.33	0.116	1

Figure 3.2: Computed user-item matrix

3.2 E-commerce product recommendation using historical purchases and clickstream data

The paper *E-commerce product recommendation using historical purchases and click-stream data* [XE18] proposed another system to generate recommendations using click-stream data. The authors use clicked and purchased items per session to create an enriched user-item matrix. They call their proposed recommender system **HPCRec**.

First of all, a **consequential table** is created by iterating through the clickstream data and mapping every session to their corresponding user, item clicks and purchases. Then a **user-item purchase frequency matrix** is created from the consequential table, which describes how frequently the user purchases an item (see Figure 3.3). The consequential table and the user-item purchase frequency matrix are the input for the actually proposed algorithm.



SessionId	UserId	Clicks	Purchases
1	1	1, 2	2
2	1	3, 5, 2, 3	2, 3
3	2	2, 1, 4	1, 2, 4
4	2	4, 4, 1, 2	2, 4, 4
5	3	1, 2, 1	1
6	3	3, 5, 2	-

consequential table

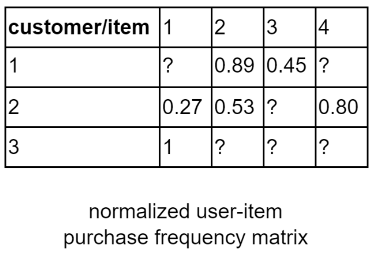
customer/item	1	2	3	4
1	?	2	1	?
2	1	2	?	3
3	1	?	?	?

user-item purchase frequency matrix

Figure 3.3: Consequential table and User-item purchase frequency matrix example [XE18]

Example: The consequential table and user-item purchase frequency matrix from Figure 3.3 will be the input. The first step is to create a normalized user-item purchase frequency matrix with the equation shown in 3.3 where X represents a row from the user-item purchases frequency matrix. The result is shown in Figure 3.4.

$$x^{'} = \frac{x}{\sqrt{x_1^2 + x_2^2 + ... + x_n^2}} \; for \; x \in X \tag{3.3}$$



customer/item	1	2	3	4
1	?	0.89	0.45	?
2	0.27	0.53	?	0.80
3	1	?	?	?

normalized user-item
purchase frequency matrix

Figure 3.4: Normalization of user-item purchases frequency matrix

Then the system iterates over the transactions **without purchases** A from the consequential table and computes the similarity between each transaction from A and each transaction **with purchases** B by comparing the clicked items of both transactions.

The comparison involves a well-known programming problem, the *longest common subsequence (LCS)*. The problem is about searching for the LCS between two sequences. HPCRec calculates an LCS rate ($LCSR$) by dividing the length of the LCS between two click sequences with the length of the longest of the two click sequences. For example for the click sequences $Session\ 6 = [3, 5, 2]$ and $Session\ 2 = [3, 5, 2, 3]$. The LCS would be $LCS = [3, 5, 2]$ and the longest sequence is *Session 2*. Then the rate would be:

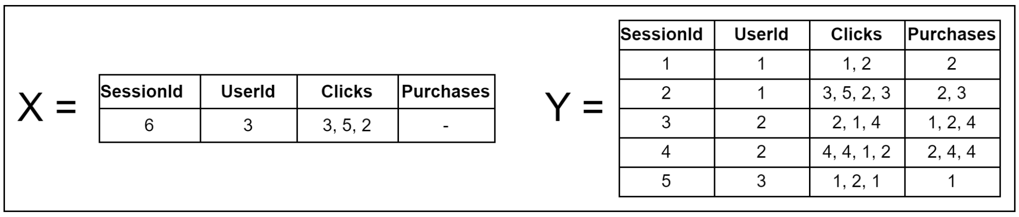
$$LCSR = \frac{|LCS|}{max(|Session\ 2|, |Session\ 6|)} = \frac{|[3, 5, 2]|}{max(|[3, 5, 2, 3]|, |[3, 5, 2]|)} = \frac{3}{4} = \underline{\underline{0.75}} \tag{3.4}$$

The rate is added by an *item frequency similarity* (IFS). The item frequency similarity between two item sequences is calculated by creating a list with distinct items from both click sequences. A vector which describes how often an item appears in the list of distinct items is created for each of the two item sequences. The IFS is the cosine distance between these two vectors. For example, the sessions from the previous examples (2 and 6) have a distinct item list of $[2, 3, 5]$. The item frequency vectors for 2 and 6 would be $6 = (1, 1, 1),\ 2 = (1, 2, 1)$. The cosine distance and the IFS would be 0.94.

LCSR and IFS have individual weights ($alpha$ and $beta$). The final formula for the similarity between two click sequences is shown in Figure 3.5 below.

$$Similarity(X, Y) = LCSR(X, Y) * alpha + IFS(X, Y) * beta \tag{3.5}$$

The similarity values are persisted into a **weighted transaction table (WTT)** together with the purchased items from transaction *B*. Transactions *A* and *B* are displayed in Figure 3.5 and the resulting weighted transaction table is displayed in Figure 3.6.



X =

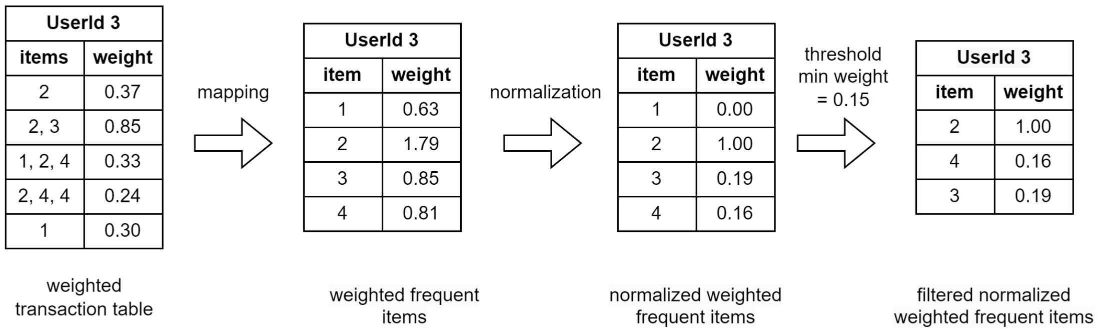
SessionId	UserId	Clicks	Purchases
6	3	3, 5, 2	-

Y =

SessionId	UserId	Clicks	Purchases
1	1	1, 2	2
2	1	3, 5, 2, 3	2, 3
3	2	2, 1, 4	1, 2, 4
4	2	4, 4, 1, 2	2, 4, 4
5	3	1, 2, 1	1

Figure 3.5: Transaction without purchases (A) and with purchases (B)

The next step is to create a list of **weighted frequent items (WFI)**, which includes all purchased items from the transactions of the **WTT**, along with a weight. The weight is based on the similarity values and it is calculated as follows: For an item, find each transaction from the table **WFI**, multiply the similarity values of the transactions by the number of occurrences of the item and sum it all. For example, the calculation for item 4 would be 0.33 + (0.24 * 2) = 0.81. The result is then used as the weight of the item. The weight is normalized afterwards and filtered through a minimum weight value threshold (e.g. 0.15). The transformation from the **WTT** to the filtered and normalized **WFI** is shown in Figure 3.6.



UserId 3	
items	weight
2	0.37
2, 3	0.85
1, 2, 4	0.33
2, 4, 4	0.24
1	0.30

weighted transaction table

UserId 3	
item	weight
1	0.63
2	1.79
3	0.85
4	0.81

weighted frequent items

UserId 3	
item	weight
1	0.00
2	1.00
3	0.19
4	0.16

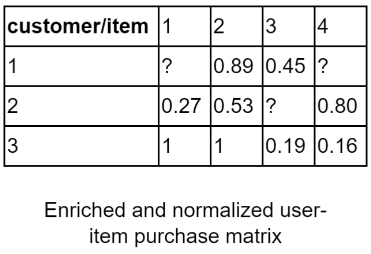
normalized weighted frequent items

UserId 3	
item	weight
2	1.00
4	0.16
3	0.19

filtered normalized weighted frequent items

Figure 3.6: Transformation: weighted transaction table to weighted frequent items

The last step is to update the normalized user-item purchase frequency matrix. Each item from the **WFI** list is updated in the user-item purchase frequency matrix for the current user of the transaction *A* if the item does not have a value yet (not null). In the example, items 2, 3 and 4 for user 3 would be updated. The resulting enriched normalized user-item purchase frequency matrix is displayed in Figure 3.7 and will be used as input for collaborative filtering.



customer/item	1	2	3	4
1	?	0.89	0.45	?
2	0.27	0.53	?	0.80
3	1	1	0.19	0.16

Enriched and normalized user-item purchase matrix

Figure 3.7: Enriched and normalized user-item purchase matrix

4 Data

4.1 novomind AG

The data used for this thesis is provided by the company novomind AG[1]. The company develops enterprise software solutions worldwide. The product portfolio ranges from information management systems[2] over digital marketplaces[3] to omnichannel contact center systems [4]. But one of the most important products is the iSHOP[5]. The iSHOP is an enterprise solution for web shops (B2B and/or B2C). The iSHOP bundles all digital commerce capabilities and modules in one box: customer management system, order management system, search engine, recommendation engine, promotion engine and more. The iSHOP provides a full back-end and front-end. It can also be operated headless with an external front-end and the back-end is easily expandable and customizable. Some well-known customers of novomind AG are *Mueller*[6], *Globetrotter*[7] and *Hagebau*[8].

The company novomind AG collects clickstream data for each webshop. This data forms the basis for this thesis. The clickstream data will be further described in section 4.2.

4.1.1 MultiChannelSelector

A web shop usually needs various forms of presentation or various sets of items, depending on the location of the shop or on the device from which the shop is accessed. Imagine an international webshop. The webshop should be fully readable and accessible for as many people as possible in any country, therefore multiple language options are required.

[1]www.novomind.com
[2]www.novomind.com/digital-commerce/ipim
[3]www.novomind.com/digital-commerce/imarket
[4]www.novomind.com/customer-service/iagent
[5]www.novomind.com/digital-commerce/ishop
[6]www.mueller.de
[7]www.globetrotter.de
[8]www.hagebau.de

Also, the laws are different in each country and some laws may forbid some items. The forbidden items must be excluded from the shop. Another use case is a webshop with multiple stores and the items vary from store to store. Some products must be excluded, according to the currently selected store.

For this purpose, novomind invented the **MultiChannelSelector** (MCS) for their web shops. The MCS consist of six dimensions:

1. Channel (e.g. web, mobile, personal computer)
2. Brand (e.g. Nike, Adidas, Razor)
3. Country (e.g. Germany, Spain, Japan)
4. Store (if the shop has multiple stores)
5. Language (e.g. German, Spanish, Japanese)
6. Currency (e.g. euro, dollar, yen)

By defining an MCS and then assigning the MCS to resources (e.g. items or categories), these resources are only accessible, if the current active MCS fits the assigned MCS. The current active MCS is usually automatically selected for a user. However, the user can indirectly change the MCS by for example changing the language or store. The dimensions have different priorities. The priority order is as listed above with channel as the highest priority and currency as the lowest priority.

The use case of the MCS can be explained with the following example:

A shop sells music albums. This shop has the following MCS:

1. (country = DE, language = DE, currency = €): The country is Germany, the language is German and the currency is euros.

2. (country = DE, language = EN, currency = €): Like above but the language is English.

3. (country = US, language = EN, currency = $): The country is USA, the language is English and the currency is dollars.

The shops are present in Germany and USA, therefore each resource has a German and English translation. In spite of that, not all music is allowed in Germany. If the user accesses the shop from Germany, then the forbidden music albums must be excluded from the shop. This can be done by not assigning the first two MCS to the music, which is forbidden in Germany. Furthermore, the currency of the USA is dollars instead of euros. The price of each music album with the third MCS will be displayed in dollars instead of euros. When a non-german-speaking user accesses the shop from Germany, then the user can change the MCS from German to English, which will change the MCS from the first one to the second one.

4.2 Clickstream data

Clickstream data is collected for every web shop which is provided by novomind. The data is collected and persisted in a Hadoop [Fou10] cluster and can be read with Impala [Fou]. If a user visits the web shop, then a session ID is created for the visit and a unique user ID (UUID) is assigned to the user. The session ID and UUID are persisted in the cookies of the user's browser. The session is ID time-bounded and resets after 30 minutes without registered activity. If the UUID can not be found in the cookies, then the UUID is reset unless the user is logged in. A registered and logged-in user always has a unique UUID. If a user interacts with the shop, then events will be triggered and these events are linked to the current session ID along with the assigned UUID. There are 18 different event types, which are described in the following section 4.2.1. Every event consists of the following properties:

Property	**Description**
channel	Describes the current channel (see 4.1.1)
brand	Describes the current brand (see 4.1.1)
country	Describes the current country (see 4.1.1)
store	Describes the current store (see 4.1.1)
language	Describes the current language (see 4.1.1)
currency	Describes the current currency (see 4.1.1)
sessionid	A unique time-bounded ID for the session
datetime	The date and time of the event
date	The date without time
year	Only the year of the date
month	Only the month of the date
UUID	A unique user ID

Table 4.1: Clickstream data common properties

4.2.1 Event types

StartSession

This event is triggered when a new session for a user is created. The event contains information about the location of the user like the **IP address** (anonymized), **latitude** and **longitude**, **region**, **city** and **postal code**. Furthermore, it contains information about the current device of the user such as **device** (e.g. Personal computer), **browser family** (e.g. Chrome), **browser producer** (e.g. Google Inc) or **operation system** (e.g. Android). Also, the event provides information about the URL of where the session starts. If the user was redirected to the shop from another website, then the redirection URL is also provided.

Information

The Information event is triggered for every new session. The purpose of this event is to detect bots by providing information about whether a JavaScript function was executed or not. Bots usually cannot execute JavaScript, so if the function was not executed, then it can be assumed that the user of the session is a bot.

Login and Logout

These events are triggered when a user logs in or logs out. The Login event provides information about the **type of user**. There are two types of users: guest and registered. A guest user is a user who wants to purchase without registering, while a registered user is fully registered. The Logout event does not provide any further information but it indicates that the user logged out.

ProductView

This event is triggered when the user opens the product page of a product. The properties of this event are the **product ID**, **item ID** and the **category ID** of the viewed product.

ContentView

This event is triggered when the user opens a page that is maintained in the back office of the shop. Every shop has a separate back office, where the shop can be managed (creation of new pages, discounts, exports, etc...).

TeaserClick and TeaserView

These two events are triggered, when a user clicks on a teaser or when a teaser is displayed. These events are currently not used as they are fired too frequently and it would cause a memory problem.

CategoryView

The *CategoryView* event is triggered when the user opens a category page. The properties of this event are the **category ID** and the **category levels** from one to four. Each category has a category path, which means each category can have a parent category and/or child category. For example, the root category *Women* could have a child category *Clothing.* The child category *Clothing* could have another child category *Shoes*, which again has a child category *Sneakers.* The category path would be Woman (Level 1) -> Clothing (Level 2) -> Shoes (Level 3) -> Sneakers (Level 4).

Search and FilterUsage

These two events are triggered when the user starts a search or/and uses filters. The Search event provides the **search term** and the **number of hits** (number of results for the search term).

The FilterUsage event describes the **filter name**, **filter value** and information about the **context of the filter usage** (filter usage during a search request or category browsing).

AddToBasket, AddToWishlist, RemoveFromBasket and RemoveFromWishlist

These events are triggered when the user adds or removes an item to or from the basket or the wishlist. The event provides the **product ID** and **item ID** of the added or removed product.

Additionally, the events AddToBasket and AddToWishlist provide a **source ID**, which reveals whichever source the item was added from (e.g. from the product page or from a quick view).

Checkout

This event is triggered at every step during the checkout process. The checkout process is the process when a user starts to purchase the items from the basket (login, view summary, payment, delivery, etc...). The steps can vary from shop to shop. The event is fired for every item that is in the basket. So if a user has n items in the basket and the checkout process has m steps, the event is fired n * m times. Each event provides information about the current step (**index**, **name** and **option** (e.g. payment type)) and about the item which is involved in the checkout process (**price**, **quantity**, **ID**, **name**, **category**).

Promotion

This event is triggered when a user uses voucher codes for an order. The event is fired for each voucher code. The event provides information about the **order ID**, **total price of the order**, **name of the voucher code** and the **voucher code** itself.

SubmitOrder

This event is triggered when the user submits an order. The event is fired for every item in the order basket, similar to the Checkout event. Also, the event provides the same item information as the Checkout event with additional information about the order (**ID** and **total price**). The event is fired for each product within the order so if a user orders n products, the submit order event is fired n times and each event contains information about one ordered product.

4.2.2 General Information and Limitations

Accuracy of the UUID

The UUID is saved as a cookie for every user who is not logged in. If a user is not logged in, changes the device or deletes the cookies, then the user will be reassigned with a new UUID. But if the user logs in, then a UUID which is linked to the account of the user is reassigned to the session. Therefore a session can consist of two UUIDs: Before login and after login. These UUIDs must be merged manually.

User path

The clickstream data does not accurately reflect the path the user takes, but rather which events the user triggers. The order of the events can be determined, but not the path the user took. For example, a user has multiple open tabs in the browser. The user triggers several events and the chronological order of the events can be determined by the timestamps. But the exact path the user took cannot be determined, because the user could interact with the shop via multiple tabs. If the user only uses one tab, then the path could be determined because there is only one point of access. However, there is no possibility to find out how many tabs the user uses and which event is fired from which tab. That problem is also illustrated in Figure 4.1.

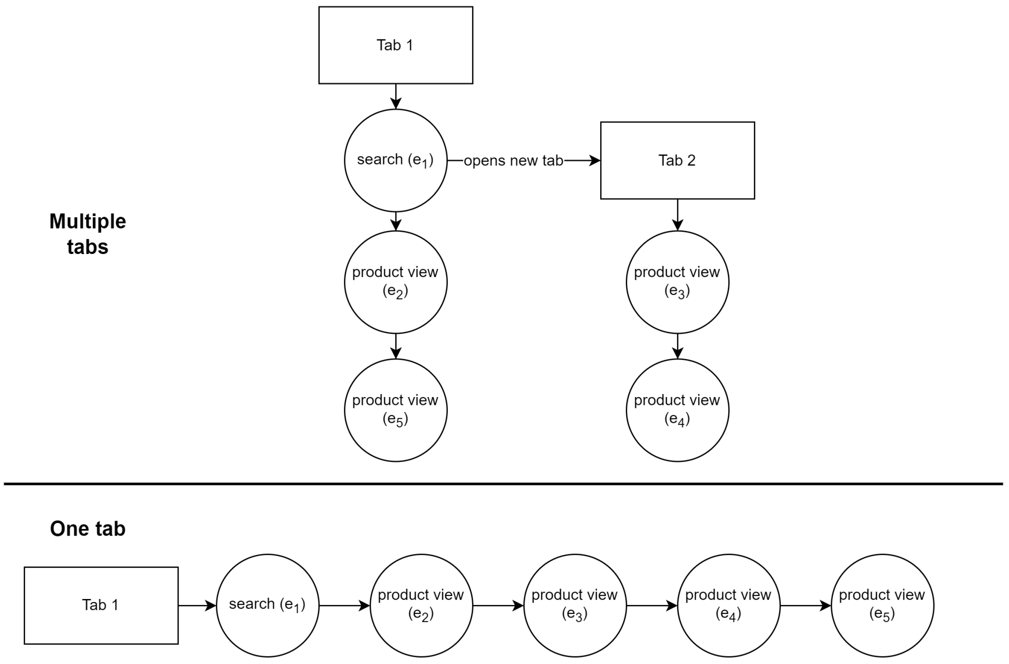


Figure 4.1: User path problem

4.3 Product data

The information about the products within the clickstream data is very limited. Fortunately, there is an API to extract current product information from a shop. The additional data is especially useful to check if a product is still valid and/or to exclude specific products. The clickstream data is historical and therefore it can hold products, which are no longer available.

The information about the products and categories is also table-structured and it consists of the following properties:

Property	**Description**
ID	The ID of the product
category path	The category path of an item (eg. Women -> Clothes -> Shoes -> Loafer), comma separated
brand	The brand of the product (e.g. Nike)
name	The name of the product
URL	The URL of the product
image URL	The image URL of the product
category ID	The category ID of the category of the product
encoded path	The category path encoded as string

Table 4.2: Product data properties

Property	**Description**
ID	The ID of the category
name	The name of the category, comma separated
level	The level of the category
path names	The category path with the name of the categories
path IDs	The category path with the IDs of the categories
prefix	MCS prefix

Table 4.3: Category data properties

4.4 Data selection

The clickstream data is persisted and managed in a Hadoop cluster. The data can be read through the query tool Impala. Unfortunately, there is no interface to access the data through the Impala engine directly. The extraction of the data is performed by logging in to a strongly secured SSH server and exporting the events (18 tables) into CSV files which then need to be downloaded.

But before the data can be extracted, the selection of the data must be done carefully, because the extraction takes a large volume of data. There are two criteria, which should be considered:

1. The shop and item range of the shop: For the evaluation, the shop should have an item range, which is specialised in one domain (e.g. fashion or outdoor equipment) or well-defined categories (e.g. multimedia or cosmetics) so that the item range can be narrowed down.
2. Period: Each shop has clickstream data for several years. This means, there are terabytes of data for each shop available. Not all of it can be used for the evaluation because it would take too long to process this large volume of data. A short period of one or two months provides already several gigabytes of data.

One of the suitable shops is **Mueller** [9]. The item range of Mueller is wide but the shop has well-defined categories which can be used to narrow down the item range. Also, some of the categories are suitable for collaborative filtering such as multimedia or toys. The ratings of items from these categories are often subjective instead of objective. A user can rate a high-quality toy negatively while another user rates it positively. Furthermore, Mueller has a high visit frequency and enough data can be extracted even within a short period of time.

The data for the evaluation of this thesis will be the clickstream data from Mueller from May 2022. Only products from the category toys which were purchased at least once will be included. In addition, all users, which did not log in (or log out) during May 2022 will be ignored. After the filtering, the data set is solid and consists of 27.977 sessions, 16.419 products and 11.510 users. The 16.419 products are viewed 130.029 times, added to the basket 84.419 times and purchased 17.655 times.

[9] www.mueller.de

5 Data transformation

5.1 Problem with the original data structure

As described in section 4.2 the clickstream data consists of 18 event types. These 18 events are distributed over 18 tables. None of these tables has an index and that leads to a very slow read performance. Another problem is, there is no information collected about which events belong to which user/session. The extraction of all events for a user or a session requires a full iteration over all 18 tables from top to bottom. Every table has millions of rows and therefore the extraction is very time-consuming. Figure 5.5 from section 5.2.2 shows, how much time it takes to retrieve events for a user with the original data structure.

In conclusion, there are two main problems with the original data structure: the missing index and the missing collected information about which events belong to which session/user. A solution for these problems will be discussed in the next section.

5.2 Transformed data structure

The first approach to solve the problems mentioned in section 5.1 was to load data into memory and then create a mapping between events, users and sessions. However, this required a high amount of RAM and it would be slow due to the large amount of data. Furthermore, the mapping would need to be done every time the algorithm starts and this would consume a lot of time. So this solution does not seem to be suitable for the evaluation. Instead of solving the problem in memory, the original data structure can be transformed into a better accessible data structure and persisted into a database.

A suitable database would be a relational database because the original data structure is already in table format. Also, relational databases provide indexing, which is required for fast-read actions. PostgreSQL version 14.4 [Gro] is selected for this thesis as it is a relational database which supports indexing. The session ID and the UUID are the most suitable columns for indexing. A table called **metadata** is created in which each session ID is mapped to the related UUID. Events for a session ID or a UUID can later be read out easily and quickly. In spite of that, the session ID and the UUID are both strings and an index for a string column uses more memory than an index for an integer column and that slows down the process.

So instead of using a string index, an artificial integer index for each event is created and added to the event. This is done by creating an integer counter and then iterating through the event tables and adding the current integer counter to the current row, while incrementing the counter each time a new index is assigned. The artificial integer index is also persisted in the *metadata* table. Every session ID is now mapped to the related UUID and the related event IDs. Then all events are persisted into a new table called *event*. The 18 event tables have common columns. These common columns can be persisted into the new table *event*. The event-specific properties are extracted into separate tables. The new table *events* will be extended with a column *type* which determines the event type.

The advantages of the new transformed structure are:

- All events are collected in one table and not in 18 tables.
- The extraction of all events for a UUID or a session ID is significantly faster because of indexing.
- The read performance is faster.
- An integer index is faster than a string index.

The disadvantages are:

- The transformation process is time-consuming.
- A join operation is required.
- Takes more memory space because two new tables and indexing are required.
- The write performance is slower.
- The join operation can not be done with a simple SQL query.

5.2.1 Illustration

The transformation can be illustrated with the following example: Figure 5.1 shows two representative event tables AddToBasket and ProductView. The red columns are common columns (these columns are present in each event table) and the blue columns are event-specific columns.

AddToBasket

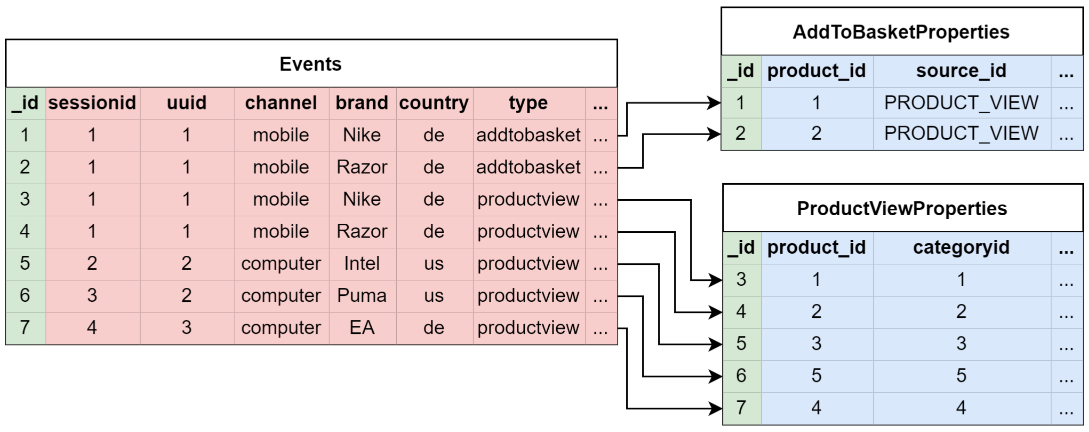
sessionid	uuid	channel	brand	country	...	product_id	source_id	...
1	1	mobile	Nike	de	...	1	PRODUCT_VIEW	...
1	1	mobile	Razor	de	...	2	PRODUCT_VIEW	...

ProductView

sessionid	uuid	channel	brand	country	...	product_id	categoryid	...
1	1	mobile	Nike	de	...	1	1	...
1	1	mobile	Razor	de	...	2	2	...
2	2	computer	Intel	us	...	3	3	...
3	2	computer	Puma	us	...	4	4	...
4	3	computer	EA	de	...	5	5	...

Figure 5.1: Original event tables: AddToBasket and ProductView

The common columns from each table can be extracted into the new table *events* and the event-specific columns are extracted into a separate table for each event type. The result is shown in Figure 5.2. The entries in the event table are assigned to an artificial ID (*_id*) and this ID is also assigned to the entry in the related property table. The type of the event is given in the new column *type*.



Events

_id	sessionid	uuid	channel	brand	country	type	...
1	1	1	mobile	Nike	de	addtobasket	...
2	1	1	mobile	Razor	de	addtobasket	...
3	1	1	mobile	Nike	de	productview	...
4	1	1	mobile	Razor	de	productview	...
5	2	2	computer	Intel	us	productview	...
6	3	2	computer	Puma	us	productview	...
7	4	3	computer	EA	de	productview	...

AddToBasketProperties

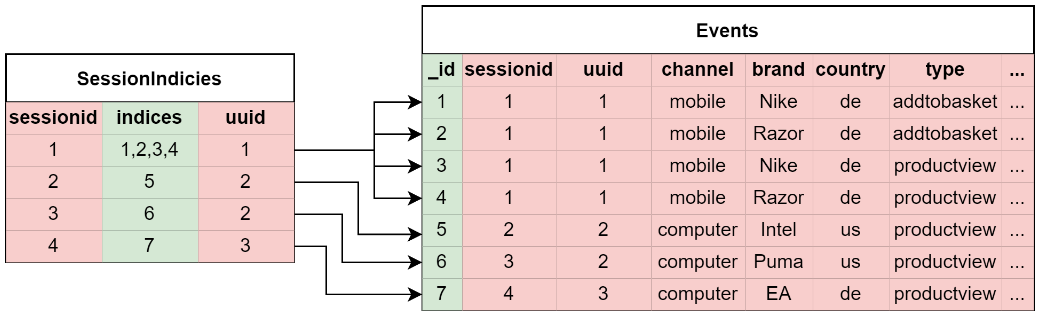
_id	product_id	source_id	...
1	1	PRODUCT_VIEW	...
2	2	PRODUCT_VIEW	...

ProductViewProperties

_id	product_id	categoryid	...
3	1	1	...
4	2	2	...
5	3	3	...
6	5	5	...
7	4	4	...

Figure 5.2: Extracted event tables: Event, AddToBasketProperties and ProductView-Properties

Finally, the last table *metadata*, which is created alongside, shows which event IDs (*_id*) and UUID belong to which session ID. This is illustrated in 5.3.



SessionIndicies

sessionid	indices	uuid
1	1,2,3,4	1
2	5	2
3	6	2
4	7	3

Events

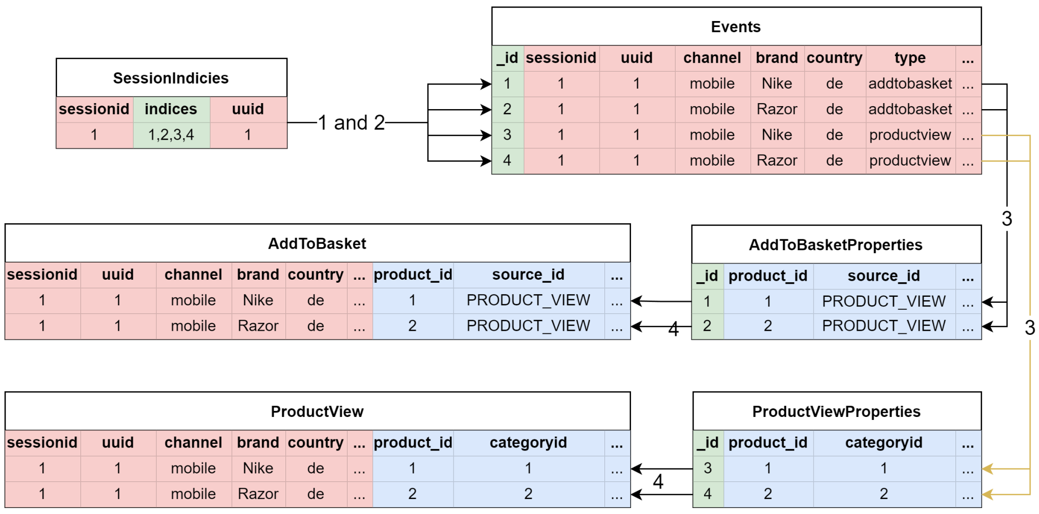
_id	sessionid	uuid	channel	brand	country	type	...
1	1	1	mobile	Nike	de	addtobasket	...
2	1	1	mobile	Razor	de	addtobasket	...
3	1	1	mobile	Nike	de	productview	...
4	1	1	mobile	Razor	de	productview	...
5	2	2	computer	Intel	us	productview	...
6	3	2	computer	Puma	us	productview	...
7	4	3	computer	EA	de	productview	...

Figure 5.3: Extracted session indices

All events for a user ID can then be extracted as follows (also illustrated in Figure 5.4):

1. Indices as string = Select the indices from the table metadata where the UUID is 1.
2. Indices as integer array = Split the indices into an array (comma is the separator) and parse them to an integer.
3. Events = For each index in integer array: Select all from events where the *_id* matches the index.
4. Properties = For each event row: Select all properties from the related type and where the *_id´s* matches.

5. Return = Join events and properties together into the original form.



SessionIndicies

sessionid	indices	uuid
1	1,2,3,4	1

Events

_id	sessionid	uuid	channel	brand	country	type	...
1	1	1	mobile	Nike	de	addtobasket	...
2	1	1	mobile	Razor	de	addtobasket	...
3	1	1	mobile	Nike	de	productview	...
4	1	1	mobile	Razor	de	productview	...

AddToBasket

sessionid	uuid	channel	brand	country	...	product_id	source_id	...
1	1	mobile	Nike	de	...	1	PRODUCT_VIEW	...
1	1	mobile	Razor	de	...	2	PRODUCT_VIEW	...

AddToBasketProperties

_id	product_id	source_id	...
1	1	PRODUCT_VIEW	...
2	2	PRODUCT_VIEW	...

ProductView

sessionid	uuid	channel	brand	country	...	product_id	categoryid	...
1	1	mobile	Nike	de	...	1	1	...
1	1	mobile	Razor	de	...	2	2	...

ProductViewProperties

_id	product_id	categoryid	...
3	1	1	...
4	2	2	...

Figure 5.4: Retrieving events for UUID 1 from the new data structure

5.2.2 Performance

The read performance with the transformed structure is by far better than the read performance with the old structure. The read execution time comparison is shown in Figure 5.5. The events for 100 users were extracted from the old structure, the transformed structure with a string index and the transformed structure with an integer index. The read execution on the integer transformed structure is approximately **50 times faster** than the read execution on the original structure. This is a significant difference. The average execution for one user for the original data structure is approx. 1.96 seconds and for the transformed data structure is approx. 0.036 seconds. Figure 5.6 shows that the integer index is more than twice as fast as the string index.

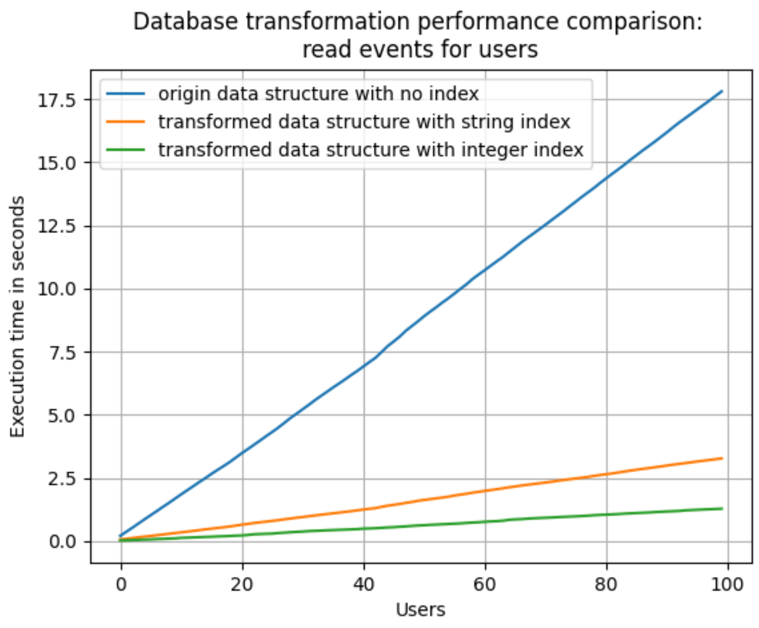


Figure 5.5: Performance comparison: read events for users

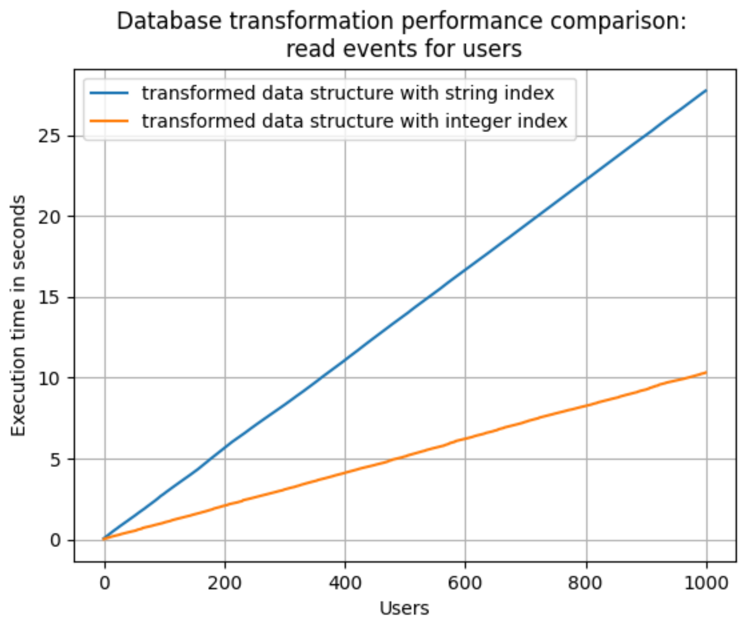


Figure 5.6: Performance comparison: read events for users

5.3 Merging the UUIDs

The UUID of a session can be ambiguous, as mentioned in section 4.2.2. If a registered user who is not logged in visits the website and the assigned UUID is not saved in the cookies, then the user receives a new UUID, until the user logs in (each account has a fixed UUID). This leads to the problem, that a session can have two UUIDs. Therefore the UUID of a session must be merged. That can be done by doing a *'login merge'*: Iterate through all events from a session and check if there is more than one UUID. If there is more than one UUID, then search for the login event and the follow-up event. The UUID of the follow-up event is correct because this UUID is linked to the account of the user. Lastly, the UUIDs from all events before the follow-up event are overwritten with the UUID of the follow-up event. Figure 5.7 illustrates this process. The graph above is the session with the two UUIDs. The UUID changes after the login event and this UUID is the correct one. The graph below is the session with the merged UUIDs. The session has now only one UUID.

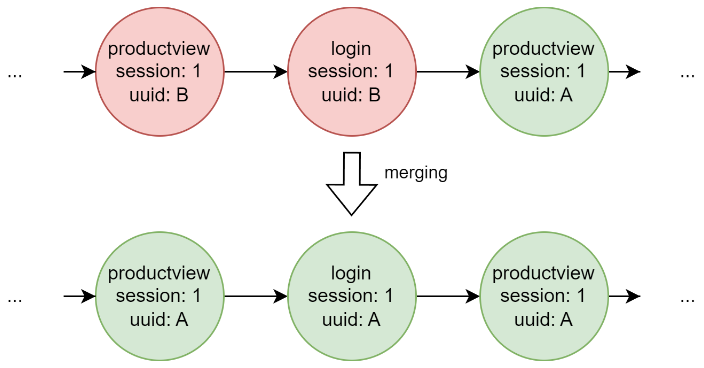


Figure 5.7: Merging of UUIDs: Example 1

Nonetheless, there are cases, where the session does not have a login event but still, the session has more than one UUID. This is an inconsistency possibly caused by a cookie blocker, cookie deletions, a failed login event trigger or other errors. These cases are relatively rare but need to be solved. It can be solved with a more complex process after the *'login merge'*: A graph can be built where a node represents a session ID or UUID. An edge connects the UUID node with the related session ID node. The graph is built by iterating over all sessions S and for each session $s \in S$ do the following:

1. Create a node for the session s.

2. Find all UUIDs U for the session s.
3. For each UUID $u \in U$:
    3.1 Create a node for the UUID u and add an edge from s to u.
    3.2 Find all sessions S_u of the UUID.
    3.3 For each session $s_u \in S_u$ repeat step 1 **if** s_u is not present in the graph.

An example is shown in Figure 5.8. The graph on the left side is the graph before the merging. There are three sessions (S_1, S_2 and S_3) and four UUIDs (U_1, U_2, U_3 and U_4). Each session has two different UUIDs ($S_1 \rightarrow (U_1, U_2)$, $S_2 \rightarrow (U_1, U_3)$ and $S_3 \rightarrow (U_3, U_4)$). It is clear to see, that the sessions are connected through the UUIDs. For the merging, the UUID node U_1 was chosen and the result of the merging is shown in the right graph.

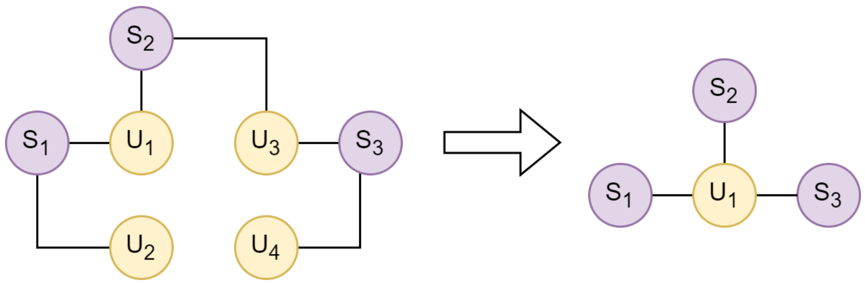


Figure 5.8: Merging of UUIDs: Example 2

6 Evaluation methods

The evaluation of a recommendation system is a difficult task as described in section 2.4. There are several points to consider such as the kind of ratings (explicit, implicit), the available budget (money and time) or/and the wanted results that should come out in the end (accuracy, diversity, coverage...). An online evaluation is excluded from this thesis because it would be too expensive and not enough time is available. Therefore two offline evaluation methods to compare the algorithms will be used to evaluate the results of this thesis:

1. Accuracy metrics: Recall, precision and F1-score.
2. Category similarity, diversity and coverage of the recommendations.

6.1 Accuracy metrics

The rating data extracted from clickstream data are usually implicit ratings. Therefore accuracy metrics are required, which are not based on explicit data. *Recall, precision* and the *F1-score* are popular metrics when it comes to explicit data. They are also used in the papers of **KimRec** [Kim+05] and **HPCRec** [XE18]. These metrics will also be used for the evaluation of this thesis. A certain percentage of relevant ratings from the user-item matrix will be removed. Each rating where the user purchased the item is assumed relevant. Then the user-item matrix with the removed ratings is used for the collaborative filtering. In the end, the resulting recommendations and the removed items for each user are used to compute *recall, precision* and the *F1-score.*

6.2 Category similarity, intra-diversity and catalog coverage and recommendation amount coverage

Accuracy metrics for recommendation systems can measure if the recommendations contain items, which the user rated positively (explicit or implicit) in the past. But this does not measure the quality of the recommendations, which is difficult to measure offline. But an approach to measure the quality offline is to compare the categories of the recommendations with the categories of the relevant items for a user. The metrics *category similarity* and *intra diversity* [Zie+05] will be used to measure the quality.

Furthermore, a recommendation system also should be able to recommend a high proportion of distinct items and it should be able to create a certain number of recommendations for each user. This can be measured and evaluated with the metrics *catalog coverage* [Jal+18] and *recommendation amount coverage.*

6.2.1 Category similarity

The similarity of two items can be calculated with their categories. An item can have one or more categories (*category set*). The *Jaccard similarity* (also known as *Jaccard coefficient* or *Jaccard index*) [Agg15] can be computed by comparing the category sets of two items. The *Jaccard similarity* calculates the number of the same elements within two sets in proportion to the number of all distinct elements. The formula (6.1) is simple: divide the size of the intersection of two sets (i_1, i_2) by the size of their union.

$$J(i_1, i_2) = \frac{|i_1 \cap i_2|}{|i_1 \cup i_2|} \tag{6.1}$$

The average category similarity between the set of recommended items REC and a set of relevant items REL to a user $u \in U$ can be computed. The first step is to map the items to their category sets $REC_u \rightarrow C_1$ and $REL_u \rightarrow C_2$. Then for each category set of C_1 calculate the *Jaccard similarity* for each category set of C_2 (equation 6.2).

$$AS_i(c_i, C_2) = \frac{\sum\limits_{c_j \in C_2} J(c_i, c_j)}{|C_2|} \quad with\ c_i \in C_1 \tag{6.2}$$

Afterwards, the average category similarity for user u (equation 6.3) is formed.

$$AS_u(C_1, C_2) = \frac{\sum\limits_{c_i \in C_1} AS_i(c_i, C_2)}{|C_1|} \tag{6.3}$$

Finally, the average of all user's average category similarity (equation 6.4) is formed.

$$AS_t(U) = \frac{\sum\limits_{u \in U} AS_u(REC_u \rightarrow C_1, REL_u \rightarrow C_2)}{|U|} \tag{6.4}$$

A high AS$_t$(U) can be interpreted as a high similarity correlation between the recommended items and the relevant items. A low AS$_t$(U) indicates a low correlation. The range of AS$_t$(U) is 0 to 1. The metric shows the similarity between the recommended and the relevant items.

6.2.2 Category intra diversity

The *intra-diversity* metric (also known as *intra-list similarity*) describes how diverse the recommended items for a user are. The higher the intra-diversity value the less diverse are the items. The intra-diversity formula proposed in [Zie+05] will be used in this thesis with small modifications to compute the intra-diversity. The first step is to calculate the intra-diversity for each user's recommendation list N. The categories are used as item properties and the similarity function will be the *Jaccard similarity.*

$$IntraDiversity_u(N) = 1 - \frac{\sum\limits_{a \in N} \sum\limits_{b \in N, a \neq b} J(a, b)}{\sum\limits_{a \in N} \sum\limits_{b \in N, a \neq b} 1} \tag{6.5}$$

The original formula uses 2 as the denominator but this does not return a value between 0 and 1 and therefore is difficult to interpret. Instead of the denominator 2, the amount of all item combinations within the recommendation list without repetition will be used. This will return a value between 0 and 1 and will make it easier to compare. Basically, the average similarity between the items within the recommendation list is calculated. Also, 1 will be subtracted so that the higher the value, the higher the diversity. Finally, the average intra-diversity will be calculated with the following formula:

$$IntraDiversity_a(U) = \frac{\sum_{u \in U} IntraDiversity_u(u \rightarrow N)}{|U|} \tag{6.6}$$

6.2.3 Catalog coverage

A good recommendation system should also cover a wide variety of items. A metric to measure the coverage of distinct items which are recommended in ratio to the total items is the *catalog coverage* [Jal+18]. The range of this metric is 0 to 1. 0 means, no item is recommended and 1 means that all items are recommended at least once. The formula is simple:

$$CatalogCoverage(R_d, N_d) = \frac{R_d}{N_d} \tag{6.7}$$

The variable R_d stands for the amount of distinct recommended items and N_d stands for the amount of distinct total items.

6.2.4 Recommendation amount coverage

A recommendation system cannot always recommend the number of items that it is supposed to recommend. A user/item could not have enough ratings to calculate a neighbourhood or there are not enough similar users/items. How capable a recommender system is to recommend a certain number of items can be measured with the following metric *recommendation amount coverage*:

$$Recommendation\ amount\ coverage = \sum_{N \in \bar{N}} \frac{|N|}{r} * \frac{1}{|\bar{N}|} \tag{6.8}$$

The metric calculates for each recommendation set N from all recommendation sets $\bar{N}$, the number of recommendations in ratio to the target amount of recommendations r. The results are added up and finally divided through the total amount of recommendation sets. The final result will be a value between 0 and 1. If the value is 0, then no recommendations are made. If the value is 1, then the targeted recommendation amount (r) is accomplished for each user.

7 Implementation

The implementation of the algorithms is made with the following tech stack.

7.1 Overview

The whole implementation for the evaluation consists of 9 python modules and one Jupyter [Klu+16] notebook:

config.py: Contains important configuration keys (see section 7.2).

database.py: Module for database interaction (execute query, read/write data frame).

transformation.py: Module for the transformation described in section 5.

object_store.py: Provides information about products and categories (see section 7.3).

extraction.py: Module for data extraction from the database (see section 7.4).

tracking_data.py: Module for the class representation of the 18 tracking data types.

kimrec.py: Module for the implementation of the *KimReco* algorithm (see section 7.5).

hpcrec.py: Module for the implementation of the *HPCRec* algorithm(see section 7.6).

collaborative_filtering.py: Module for the collaborative filtering (see section 7.7).

evaluation.py: Module for the evaluation methods (see section 7.8).

collaborative_filtering_experiment.ipynb: Jupyter notebook for the experiment (see section 8).

7.2 Configuration (*config.py*)

The configuration of the evaluation parameters can be controlled over several configuration keys. These keys are defined in the module *config.py*. The following table 7.1 describes the configuration keys:

Configuration key	**Description**
PRODUCT_DATA_PATH	The path of the additional product information location
EVENT_DATA_PATH	The path of the raw event data location (CSV files). Required for the transformation process.
TRANSFORMATION_CHUNK_SIZE	Chunk size for the transformation.
POSTGRES_USERNAME	The user name of the PostgreSQL database.
POSTGRES_PASSWORD	The password for the user *POSTGRES_-USERNAME.*
POSTGRES_HOST	The host of the PostgreSQL database.
POSTGRES_DATABASE	The name of the PostgreSQL database.
POSTGRES_ACTIVE	Boolean configuration key which indicates, if the database is active. The database is only required for the transformation and preprocessing but not for collaborative filtering.
OBJECT_STORE_MODE	Mode of the object store (0 = database or 1 = RAM). See section 7.3.

Table 7.1: Configuration keys

7.3 Object store (*object_ store.py*)

The additional information about the products, which is described in section 4.3, must be retrievable for each product ID. Therefore an object store was implemented as an object, which holds the additional information about the products. The main task of it is to search the additional information for a product ID and then return the additional information. The implementation is located in the module *object_ store.py* and the class diagram for the object store is shown in Figure 7.1 below.

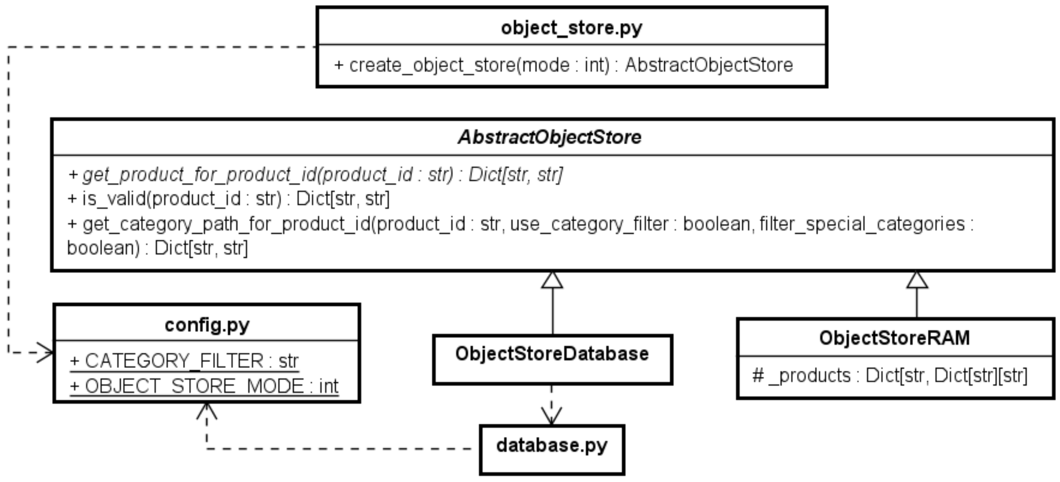

object_store.py
+ create_object_store(mode : int) : AbstractObjectStore
AbstractObjectStore
+ get_product_for_product_id(product_id : str) : Dict[str, str]
+ is_valid(product_id : str) : Dict[str, str]
+ get_category_path_for_product_id(product_id : str, use_category_filter : boolean, filter_special_categories : boolean) : Dict[str, str]
config.py
+ CATEGORY_FILTER : str
+ OBJECT_STORE_MODE : int
ObjectStoreDatabase
ObjectStoreRAM
# _products : Dict[str, Dict[str][str]
database.py

Figure 7.1: UML class diagram of the object_store.py module

There is one abstract object store (*AbstractObjectStore*) and two implementations (*ObjectStoreDatabase* and *ObjectStoreRAM*). The abstract object store provides two functions with default implementations and one abstract function. The abstract function is *get_product_for_product_id* and the task of this function is to extract the additional product information for a product ID. The default functions are:

get_category_path_for_product_id: Retrieves the category path for a product ID. The optional input parameter *use_ category_filter* is a boolean and if it is set to true, then the global category filter, which is defined in the *config.py*, is used to include categories, which are children of the category filter (default = true). The other optional input parameter *filter_ special_ categories* is also a boolean and is used to filter out all special categories which are marked by the prefix '*-s-*' (default = true). The function calls the abstract function *get_product_for_product_ id.*

is_valid: Checks if a product is valid. The criteria for validation are:

1. If the *category_filter* is not null (resp. None), then the product must contain the category ID given by the *category_filter*,

2. The additional information about the product must be present. If it is not present, then the product is outdated and thus no longer relevant and therefore not valid.

The function calls the abstract function *get_ category_ path_ for_ product_ id*.

There are two implementations for the abstract object store: *ObjectStoreDatabase* and *ObjectStoreRAM*. The *ObjectStoreDatabase* persists the additional data in the database and then reads all information out of the database. The *ObjectStoreRAM* persists the additional product information into the RAM. The *ObjectStoreDatabase* is mostly used for debugging purposes because it is fast initialized but slow in execution. The *ObjectStoreRAM* is faster in execution but needs more time to initialize and the larger the additional product information is, the slower the execution.

7.4 Extraction (*extraction.py*)

The task of the extraction module is to extract data from the database. The UML diagram in Figure 7.2 shows the functions of the module.

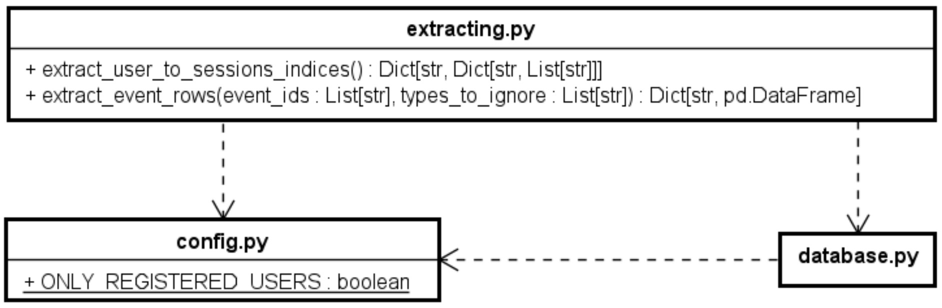

extracting.py
+ extract_user_to_sessions_indices() : Dict[str, Dict[str, List[str]]]
+ extract_event_rows(event_ids : List[str], types_to_ignore : List[str]) : Dict[str, pd.DataFrame]
config.py
+ ONLY_REGISTERED_USERS : boolean
database.py

Figure 7.2: UML diagram of the extraction.py module

The first function is called *extract_ user_ to_ sessions_ indices* and it extracts all UUIDs and their corresponding session IDs and event IDs. The function uses the *metadata* table (see section 5) to extract the UUIDs, session IDs and event IDs. If the configuration key *ONLY_ REGISTERED_ USERS* is set to true, then unregistered users are excluded. A registered user is a user who triggered a login or logout event at least once.

The second function is called *extract_event_rows* and is used to extract all event rows in form of data frames from the database for specific event IDs. The function reads and joins the event and property tables as described in section 5. The function does not need any further inputs besides the event IDs, but an optional input *types_to_ignore* can be passed, which can be used to exclude specific event types.

7.5 KimReco (*kimrec.py*)

The algorithm *KimReco* from the paper [Kim+05] was described in section 3.1. The algorithm extracts behavioral and navigational patterns from clickstream data and then creates a user-item matrix from it. This algorithm is adapted and implemented for the clickstream data from novomind. The adaptation and implementation will be described in the following sections.

7.5.1 Selection and extraction of features

Seven features were extracted in the original algorithm: click type, view time, number of visits, level 1 and 2 category ratio, basket placement and purchase. 2 of these 7 features cannot be extracted from the novomind clickstream data: click type and view time. This is because of the user path problem, which was discussed in section 5.1. Click type requires knowledge about the previous event and the view time requires information about the next event:

$$view\ time\ of\ product\ p\ from\ event\ e = timestamp\ of\ \ e + 1 - timestamp\ of\ \ e \tag{7.1}$$

Due to the user path being ambiguous, these features cannot be reliably extracted. Therefore the features click type and view time will be excluded in the implementation for this experiment. However, the variable view time will be replaced with a new variable called *first attention time*. This variable describes how much time the user needs to trigger another event after the user viewed a product. If the *productview* event is the last event of a session, then the average of all attention times is set as a default value.

The features *number of visits*, *category level 1 and 2 ratios*, *basket placement status* and *purchase status* can be extracted without any problems. The number of visits is calculated by counting the *productview* events for each user and product. The basket placement status can be determined by using the *addtobasket* event and the purchase status can be determined by using the *submitorder* event. There are cases, where a user has not viewed a product (this means there is no *productview* event) but there is an *addtobasket* or *submitorder* event. For these cases, the *addtobasket* and *submitorder* events are counted as one product click.

The category ratios level 1 and 2 for a product from a user can be calculated by comparing the category of the product with the categories of all products the user viewed. The formula in Figure 7.2 shows how the category ratio is calculated. P represents the viewed products of a user, l stands for the category level and cp for category path.

$$category\ ratio(p_a, l) = (\sum_{p_b \epsilon P} \begin{cases} 1, & \text{if } cp[l]\ of\ p_a\ \epsilon\ cp\ of\ p_b \\ 0, & \text{otherwise} \end{cases}) * \frac{1}{|P|} \tag{7.2}$$

$$with\ p_a \in P$$

7.5.2 Selection of machine learning algorithm

KimRec uses the extracted features to estimate the basket placement status for all items of a user, which were clicked but not added to the basket. The (estimated) basket placement status and the purchase status are then used to calculate the ratings for the user-item matrix. Machine learning is used for the estimation of the basket placement status and in the original paper [Kim+05] decision trees, an artificial neural network and logistic regression are used, whereby logistic regression performed the best. Another paper from New Delhi [RGS17] also tested gradient boosting and random forest for the basket placement estimation. Gradient boosting and random forest provided better overall results and the authors have demonstrated that there might be better algorithms for the basket placement estimation.

Unfortunately, the authors did not compare the performance of the machine learning algorithms within the basket placement estimation, but they compared the performance in the context of the full collaborative filtering process. While this makes sense as the final result is the interesting object, the performance of the machine learning algorithms would also be worth knowing. Therefore, for the implementation of the KimRec for this thesis, the basket placement estimation is tested with six machine learning algorithms and their performance was compared afterwards. The algorithms were: *gradient boosting*, *AdaBoost*, *random forest*, *decision trees*, *logistic regression* and *artificial neural network (multilayer perceptron)*. The data is split into a training set (70% of the data) and a test set (30% of the data). The pre-implemented machine learning algorithms from the python library *scikit-learn* [Ped+11] were used for the test. Nine different sets of training and test data were used for each algorithm and the results were averaged. The results of the test are shown in the graph 7.3 below.

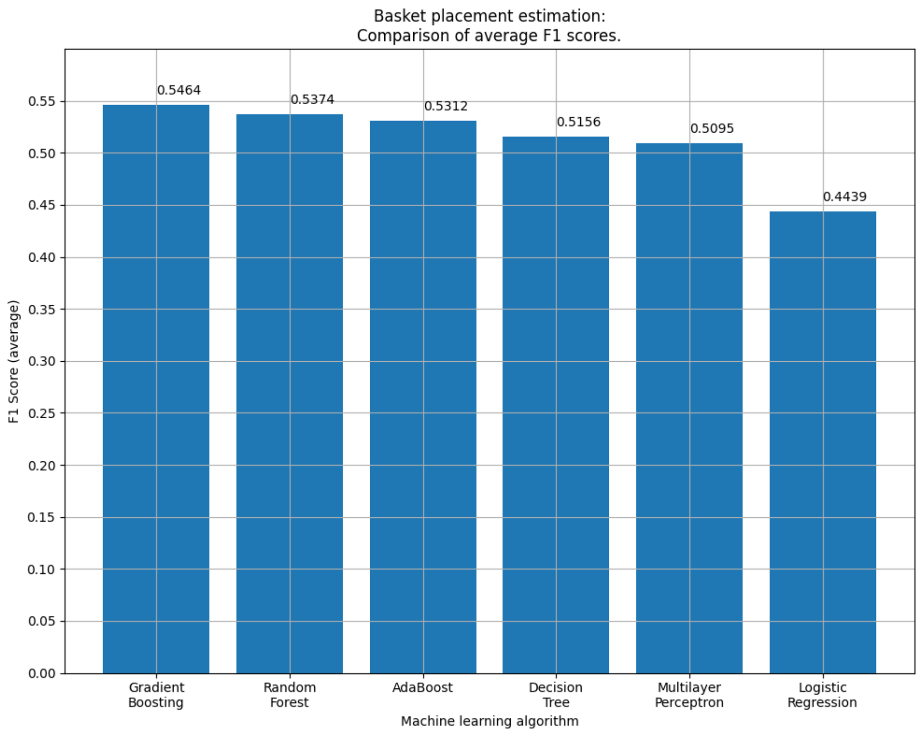


Figure 7.3: Basket placement estimation. Comparison of average F1 scores.

Gradient boosting produced the best results with the highest average F1 score. But the scores of random forest, Ada boost, decision trees and the multilayer perceptron are also close to the score of gradient boosting. By far the worst algorithm is logistic regression with a low average F1 score of 0.4439. This comes as a surprise because in the original paper [Kim+05] logistic regression performed very well. Since **gradient boosting** achieved the highest F1-Score, it is chosen for the implementation.

7.5.3 Final Implementation

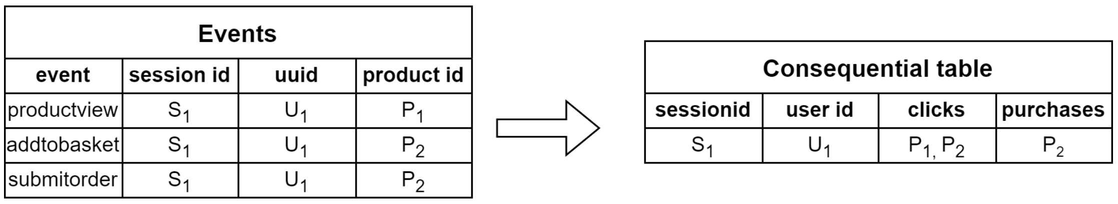
After the selection of the features and the machine learning algorithm for the basket placement estimation, the final implementation is made by following the procedure as described in section 3.1. The first step is to extract all users and their corresponding event IDs from the database with the *extraction.py* module. Then iterate over the events for each user and create the behavioral data set for every user. The behavioral data set for a user consists of all products which the user interacted with and their extracted features. After all behavioral data sets are created, the sets are merged into one large data frame. Then the missing attention time is computed by forming an average over the available attention times. Afterwards, the missing attention time values are replaced with the average value. The next step is to estimate the basket placement status with gradient boosting. Finally, the user-item matrix is created.

7.6 HPCRec (*hpcrec.py*)

The algorithm **HPCRec** from the paper [XE18] was described in section 3.2. The algorithm extracts the user ID, product clicks and product purchases for every session into a *consequential table*. The *consequential table* serves as the basis for an *item-user purchase matrix*, which describes how often a user purchases an item. The *consequential table* and the *item-user purchase matrix* are the input for the actual HPCRec algorithm, which returns an ***enriched*** *item-user purchase matrix*. The enriched matrix serves as the item-user matrix input for collaborative filtering.

7.6.1 Consequential table

The first step to create the *consequential table* is to extract all session IDs and their related UUID and event IDs with the *extraction.py* module. Then for each session, which at least contains one **valid** item, iterate over all events and add every item from the *productview* and *addtobasket* event to the click list along with all items from the *submitorder* to the purchase list. The session ID, user ID, click list and purchase list are then added as a row to the consequential table. An example of a session transformed to a row is illustrated in Figure 7.4 below.



Events

event	session id	uuid	product id
productview	S_1	U_1	P_1
addtobasket	S_1	U_1	P_2
submitorder	S_1	U_1	P_2

Consequential table

sessionid	user id	clicks	purchases
S_1	U_1	P_1, P_2	P_2

Figure 7.4: Session to *consequential table* example

There is a special case: A session has a time-bounded lifespan of 30 minutes after the last event and if a user is passive for too long, then a new session will be created. This can lead to a situation, where two or more sessions are logically one process. Chains of sessions for one user exist, where the user just views and adds items to the basket and then in the last session only submits an order. The problem is, the session with the *submitorder* event does not have any *productview* nor *addtobasket*. Therefore, the click list for this session would be empty and an empty click list cannot be used for the HPCRec algorithm. The fallback solution for this special case is to look back on all sessions before the session with the *submitorder* event, find the session where the *productview* and the *addtobasket* events match the purchases and merge these sessions.

Nevertheless, the fallback solution is not fully accurate, but it protects valuable data against loss. In the future, a deeper inspection of this problem is necessary to reconstruct processes from a chain of sessions.

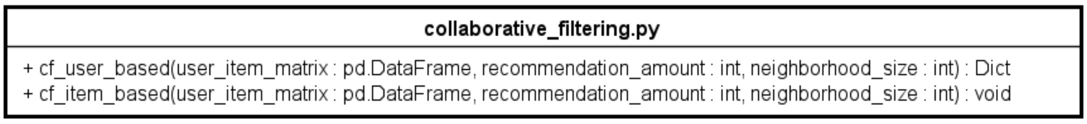
After the *consequential table* is created, the *item-user purchase matrix* can be created. The actual HPCRec algorithm is implemented as described in section 3.2, yet there are two pieces of information, which are noteworthy:

1. If the similarity value between a session without purchases and a session with purchases is zero, then it will not be added to the *weighted transaction table*, because their significance is not given.
2. The calculation of the longest common subsequence, a well-known programming problem, is involved in the calculation of the similarities. There are several solutions to this problem but two of the most popular approaches are the recursive approach and the dynamic programming approach. The basic recursive approach uses less memory but is slower, while the dynamic programming approach is faster but requires more memory. If there is enough memory for the dynamic programming solution, it will perform better in general. The click sequences are not very large

and there is enough memory available, so the dynamic programming approach is the best fit.

7.7 Collaborative filtering (*collaborative_filtering.py*)

The user-based and item-based collaborative filtering techniques are implemented in the module *collaborative_filtering.py*. The implementation follows the description described in 2.3.2. The cosine similarity is used as the similarity function. The diagram 7.5 shows a small overview of the module:



collaborative_filtering.py

+ cf_user_based(user_item_matrix : pd.DataFrame, recommendation_amount : int, neighborhood_size : int) : Dict
+ cf_item_based(user_item_matrix : pd.DataFrame, recommendation_amount : int, neighborhood_size : int) : void

Figure 7.5: UML diagram of the collaborative filtering.py module

The function *cf_user_based* implements user-based collaborative filtering, while *cf_-item_based* implements item-based collaborative filtering. Both functions have the same input parameter:

user_item_matrix: Contains ratings for the user and items in form of a matrix. The input type is a data frame because the output of the KimRec and the HPCRec is a data frame.

recommendation_amount: Number of recommendations to be generated.

neighborhood_size: Size of the neighborhood of user/items.

The functions are optimized by parsing the user-item matrix from a data frame to a Numpy array. Numpy uses C in the background and C is faster than pure Python. So by using only Numpy arrays, the collaborative filtering execution is faster. Furthermore, before calculating the similarities and preferences, all rating indices of a user or item are mapped into a dictionary. Instead of iterating through the whole matrix, these rating indices help to directly access the relevant cells. This improves the performance in terms of execution time. The last optimization is that the similarity between users is only calculated if the users have at least one common rating otherwise the similarity will always be zero.

7.8 Evaluation (*evaluation.py*)

The *evaluation.py* module realizes the implementation of the evaluation methods (see section 6). The class diagram in Figure 7.6 shows the structure of the module:

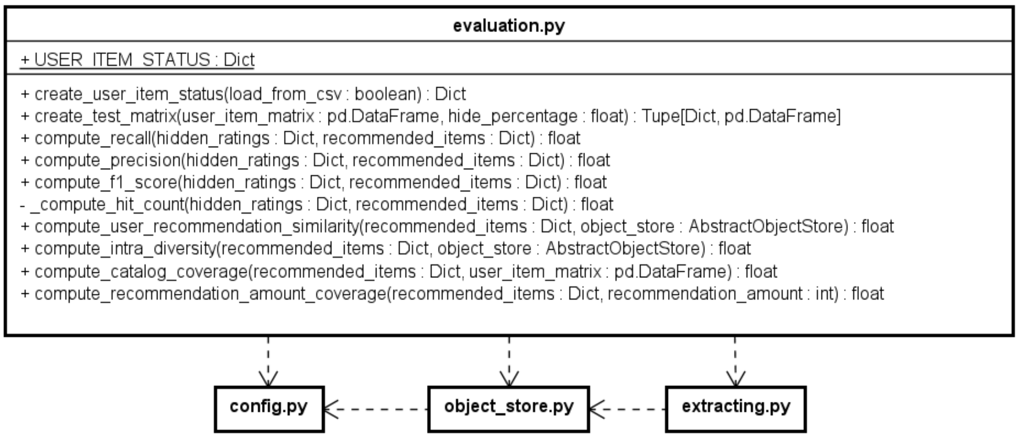

evaluation.py
+ USER_ITEM_STATUS : Dict
+ create_user_item_status(load_from_csv : boolean) : Dict
+ create_test_matrix(user_item_matrix : pd.DataFrame, hide_percentage : float) : Tupe[Dict, pd.DataFrame]
+ compute_recall(hidden_ratings : Dict, recommended_items : Dict) : float
+ compute_precision(hidden_ratings : Dict, recommended_items : Dict) : float
+ compute_f1_score(hidden_ratings : Dict, recommended_items : Dict) : float
- _compute_hit_count(hidden_ratings : Dict, recommended_items : Dict) : float
+ compute_user_recommendation_similarity(recommended_items : Dict, object_store : AbstractObjectStore) : float
+ compute_intra_diversity(recommended_items : Dict, object_store : AbstractObjectStore) : float
+ compute_catalog_coverage(recommended_items : Dict, user_item_matrix : pd.DataFrame) : float
+ compute_recommendation_amount_coverage(recommended_items : Dict, recommendation_amount : int) : float
config.py
object_store.py
extracting.py

Figure 7.6: UML diagram of the evaluation.py module

The global static dictionary *USER_ITEM_STATUS* contains for each user the viewed, added to basket and purchased items. The variable is used to find out which items a user purchased. The variable is initialized by iterating through the productview, addto-basket and submitorder events. An item in productview is marked as *viewed*, an item in addtobasket is marked as *basket* and an item in the submitorder event is marked as *purchased*. The items can be assigned to the users via the UUID of the events. The *USER_ITEM_STATUS* can alternatively be created from a pre-computed JSON file.

The task of the function *create_test_matrix* is to hide a certain amount of ratings of items which were purchased by the related user. First, the function *get_items_to_hide* finds all rating candidates. Then the function takes a certain percentage (*hide_percentage*) of the candidates which will then be the ratings to be hidden. The candidates are chosen randomly and if a user has only two ratings left, then the ratings of this user will not be considered as a candidate. That means, at the end each user will have at least two ratings. The functions return the hidden ratings as a dictionary (user ID → hidden item IDs) and the new user-item matrix with the hidden ratings.

There are functions for each evaluation metric (*recall*, *precision*, *F1-score*, *category similarity*, *intra diversity*, *coverage* and *recommendation amount coverage* - see section 6).

The object store in the functions *compute_ user_ recommendation_ similarity* and *compute_ intra_ diversity* is required to get the categories of the items.

8 Setup and Results

8.1 Setup

The test objects are the algorithms **KimRec**, **HPCRec** and the **user-item matrix**, which both algorithms produced. **Three test and validation sets** are created for each user-item matrix. Each test set will be used to produce recommendations with **user-based** and **item-based** collaborative filtering. The **neighborhood size** will be *3, 5 and 10* and the **recommendation amount** will be *5, 10, 20 and 30.* Each combination of the test object algorithms, collaborative filtering algorithm, neighbourhood size and recommendation amount will be tested on the three test and validation sets and then the results will be averaged later.

The experiment will be executed on a Jupyter notebook [Klu+16] and the technical information of the Jupyter servers are:

System: Linux, 123-Ubuntu SMP

Processor: x86_64

Physical cores: 6

Total cores: 12

Core frequency: 800.00 Mhz - 4600.00 Mhz

RAM: 62.75 GB

Disc space: $\approx$ 467 GB

8.2 Results

The results of the evaluation are discussed in the following sections with the help of graphs. Each of the graphs describes a metric value for the **KimRec** (blue) and **HPCRec** (red) algorithm, whereby the y-axis describes the metric value and the x-axis describes the recommendation amount. The neighbourhood sizes are described by the different line types: a solid line stands for a neighbourhood size of 3, a dashed line for 5 and a dotted line for 10. Two graphs are plotted per metric: one graph for user-based and one graph for item-based. This makes the comparison clearer and easier.

8.2.1 Accuracy metrics

Recall

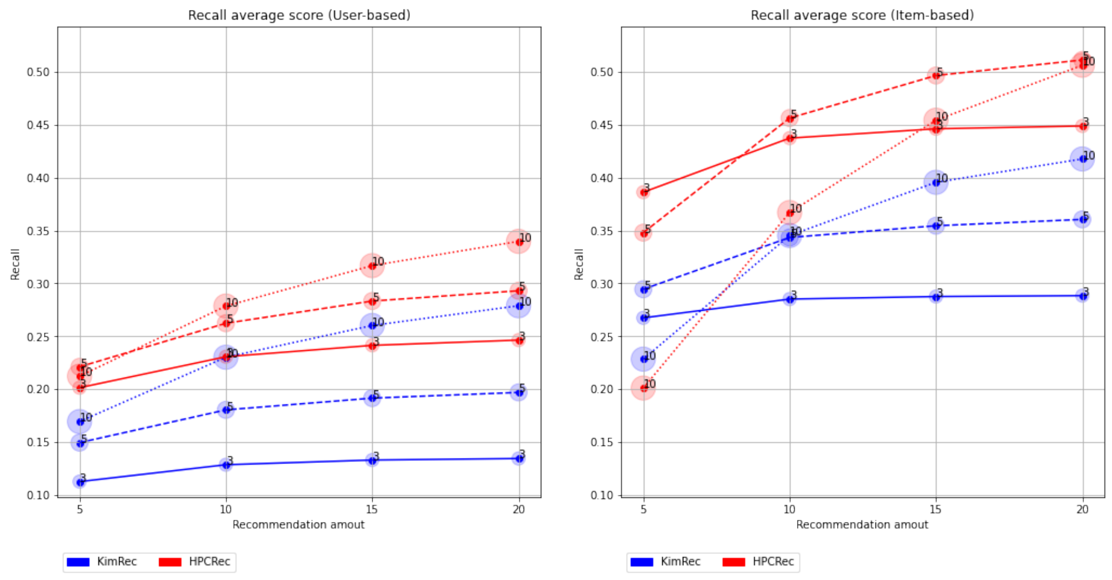


Figure 8.1: Average recall score for user-based and item-based collaborative filtering

The graphs in Figure 8.1 show the recall for user-based and item-based collaborative filtering. Recall describes how many relevant items were found in relation to the number of hidden items. HPCRec is mostly superior in both techniques. In total HPCRec is more capable of re-recommending relevant items to users.

Item-based performs significantly better than user-based. A reason for this could be that the item rating vectors are less sparse than the user rating vectors because a user rates fewer items compared to an item being rated. Therefore the similarity calculation in item-based could be more precise.

A small neighbourhood size seems to be better at finding relevant items because a small neighbourhood will lead to a small **recommendation pool** (all items a user did not rate, but his neighbours did). A small recommendation pool will be less diverse because it contains rated items from the top n similar neighbours. Therefore, the chance of finding an item relevant to a user is higher, but the recommendations might be too overfitting.

The recall with a large neighbourhood size increases more strongly together with the number of recommendations because the recommendation pool is larger and as more items are recommended the higher the chance to find relevant items.

Precision

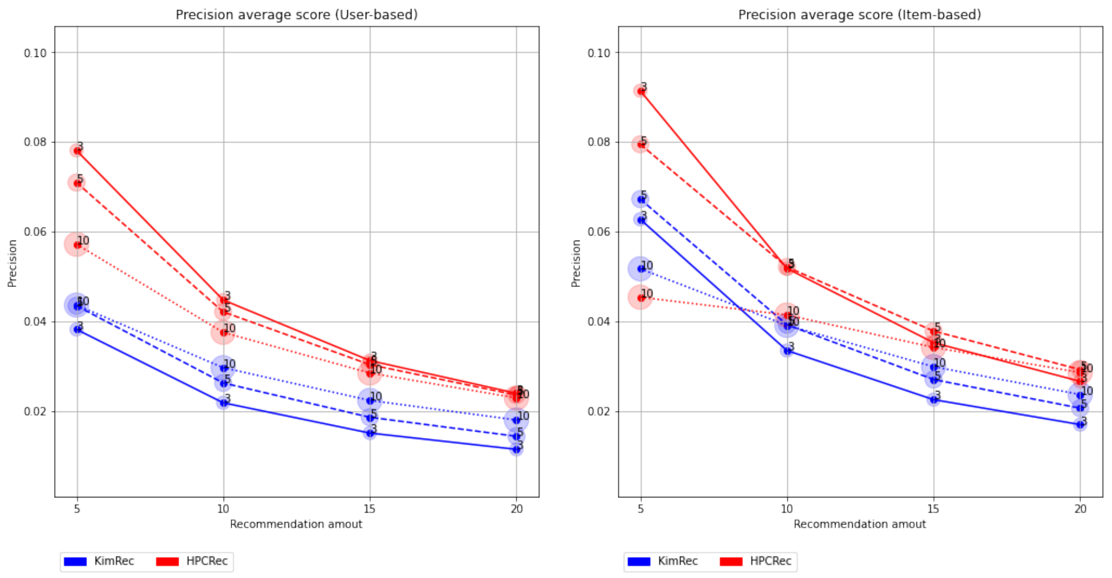


Figure 8.2: Average precision score for user-based and item-based collaborative filtering

Figure 8.2 above shows the precision graph, the counterpart of recall. Precision describes how many relevant items were found in relation to the number of total recommendations. The precision is low in general for both user-based and item-based because the amount of hidden items is low and therefore it is very unlikely that a recommended item for a user matches with one of the hidden items of this user. Also, if the number of recommendations increases, then the precision decreases because more items are being recommended, where it does not match the relevant items.

Item-based has slightly better precision than user-based. HPCRec has better precision, especially with user-based collaborative filtering, where the precision is higher at any recommendation amount.

F1-Score

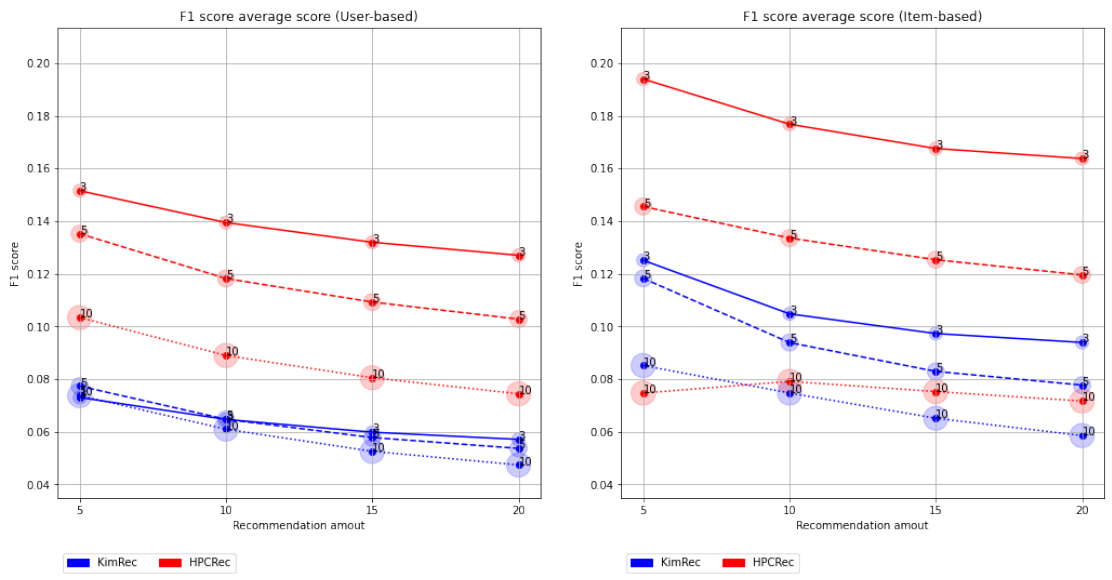


Figure 8.3: Average F1-score for user-based and item-based collaborative filtering

The F1-score aims to balance recall and precision to fit both metrics into one metric. The F1-score results are shown in Figure 8.3. Item-based already has a better recall and precision score and therefore the F1-score for item-based is in total higher than user-based. Clearly, HPCRec, which already has a better recall and precision, has a higher score than KimRec.

If one only refers to the accuracy metric recall, precision and F1-score, then the HPCRec algorithm with **item-based** collaborative filtering performs best.

8.2.2 Intra diversity

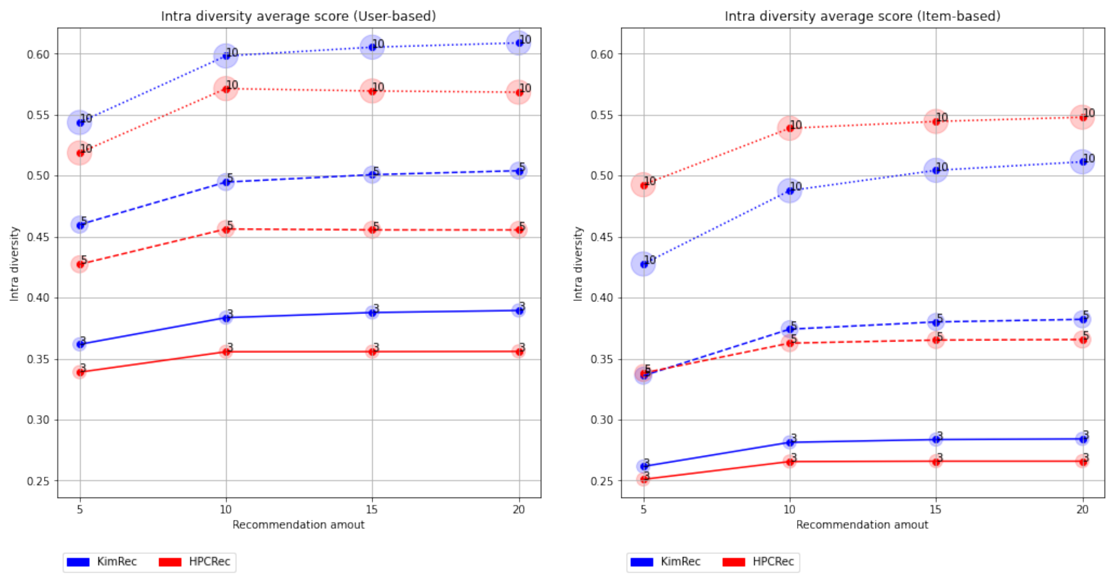


Figure 8.4: Average intra-diversity score for user-based and item-based collaborative filtering

Intra-diversity describes how different the items are within the recommendations for a user and the result for the intra-diversity is shown in Figure 8.4. A high score indicates a high variation of items.

Evidently, a large neighbourhood and a large number of recommendations are correlated to high diversity. If the neighbourhood is larger, then the users are more likely to be mixed. If the neighbourhood is more varied, then the possible amount of recommendations will be larger. Therefore the items provide more variation and the recommendations will differentiate more.

The intra-diversity of **KimRec** and **HPCRec** are on a similar level and there is not a big gap between them. KimRec is slightly better with user-based while HPCRec is slightly better with item-based. Therefore, it is difficult to say, which one generates the most diverse recommendation. Although, item-based generates on average more diverse recommendations.

8.2.3 Category similarity

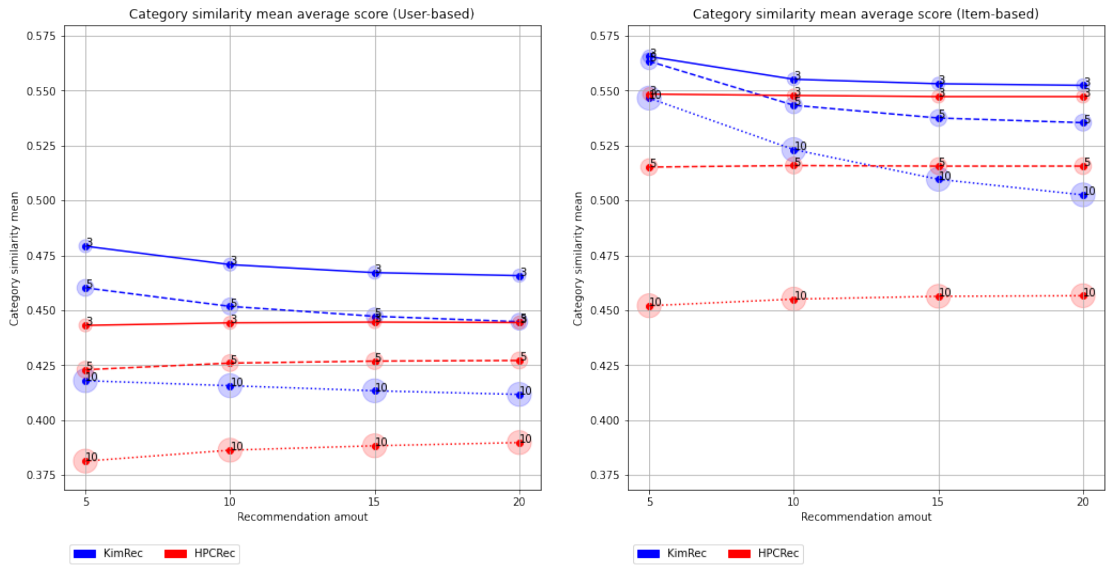


Figure 8.5: Average category similarity score for user-based and item-based collaborative filtering

Category similarity describes how similar the recommended items to the purchased items of a user are, based on the categories. It is not to be confused with the intra-diversity because the intra-diversity only shows how different the items within the recommendations are. The target of the category similarity is to measure how the recommendations fit the already purchased items of a user. If the value is too high, then the recommendations may be too similar and the user does not receive item recommendations out of his normal preferences. If the value is too low, then the recommendation could be too different. A balanced value of around 0.5 shows that the recommendations still fit the preferences of the user and still display variety.

The graphs in Figure 8.5 show the category similarity for the user-based and item-based techniques. User-based tends to be a bit more unfitting as item-based tends to be a bit more overfitting. However, both techniques are close to 0.5 and therefore acceptable. The user-item matrix from the KimRec provides more similar items than HPCRec in total. Furthermore, it seems the neighbourhood size and the number of recommendations do not have a significant influence on the category similarity.

8.2.4 Catalog coverage

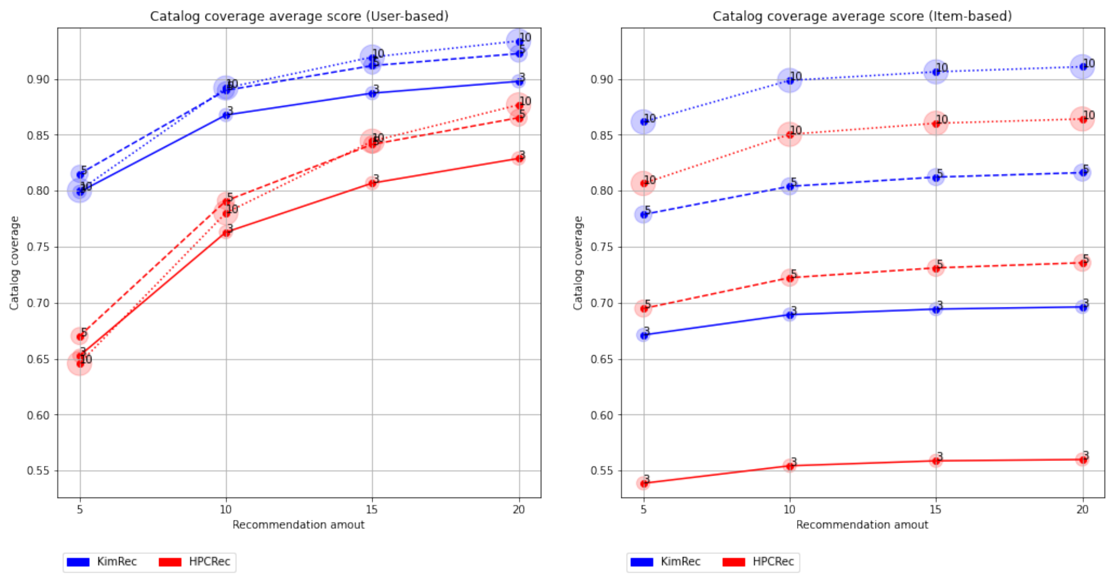


Figure 8.6: Average catalog coverage score for user-based and item-based collaborative filtering

Catelog coverage describes how many distinct items were included in the recommendations. If the coverage is 1, then all items are included. Coverage of 0 shows that no items are included. The graphs in Figure 8.6 show the result of the catelog coverage evaluation.

User-based can cover more items on average than item-based. As the number of recommendations increases, the coverage in user-based increases more strongly than in item-based. Naturally, the larger the number of recommendations, the higher are the chances of including new items. A small neighbourhood seems to result in the lowest coverage while a larger neighbourhood results in the highest coverage. That is logical as the larger the neighbourhood is the more items are in the recommendation pool. KimRec covers more items than HPCRec, especially with the user-based technique.

8.2.5 Recommendation amount coverage

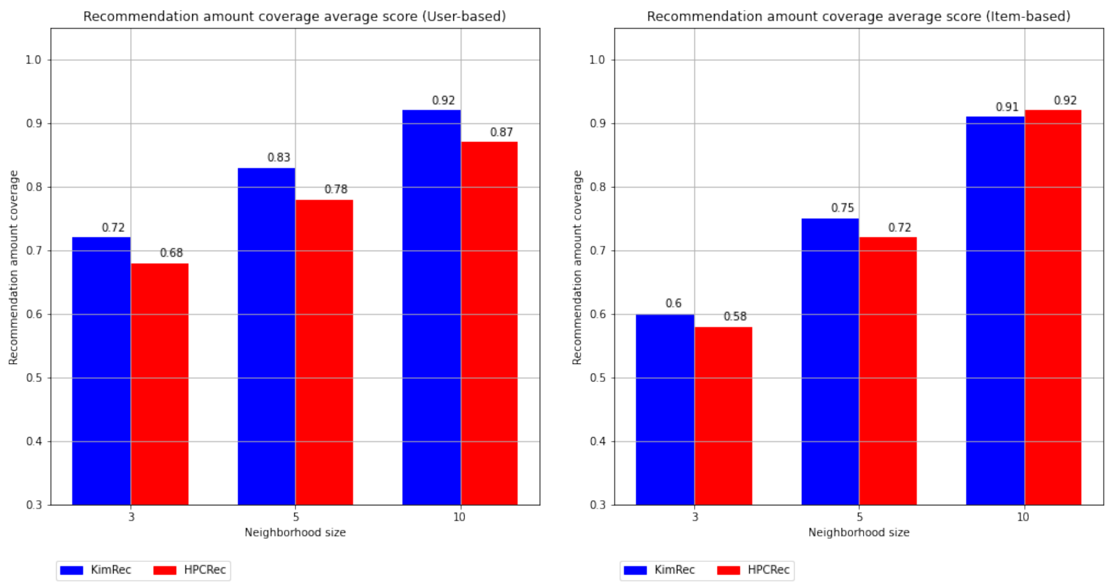


Figure 8.7: Average recommendation amount coverage score for user-based and item-based collaborative filtering

Recommendation amount coverage shows how capable a recommendation system is to generate the intended number of recommendations. If a recommendation system should generate 10 recommendations for a user but is only capable to generate 7, then the recommendation amount coverage is 0.7.

As shown in the graphs from Figure 8.7, user-based is less capable to generate the intended number of recommendations. The maximum coverage in user-based is 0.92 and that is only if the neighbourhood is 10. Item-based exceeds the coverage of 0.92 with a neighbourhood size of 5 and almost maxed out with maximum coverage of 0.99. Therefore item-based is better in this aspect.

KimRec is slightly better than HPCRec but the difference decreases as the neighbourhood size increases.

8.3 Summary

The evaluation results are summarized in table 8.1, where the collaborative filtering technique with KimRec and HPCRec are ranked for each metric. The highest rank is 4 and the lowest rank is 1. The results give several information:

1. Item-based is superior to user-based in almost every aspect except the catelog coverage.
2. HPCRec is superior in terms of accuracy.
3. KimRec is superior in terms of coverage as it can recommend more similar items.

Metric	**KimRec**		**HPCRec**	
CF technique	**User-based**	**Item-based**	**User-based**	**Item-based**
Recall	1	3	2	4
Precision	1	3	2	4
Intra-diversity	1	2	1	2
Category similarity	2	4	1	3
Catalog coverage	4	3	2	1
Recommendation amount coverage	2	4	1	3
Summary	*12*	*20*	*9*	*19*

Table 8.1: Direct comparison

Although KimRec is less accurate, it shows its strength in other aspects. Another advantage of KimRec is that the preprocessing is faster than for HPCRec. However, HPCRec is more accurate and it produces more diverse results due to the extension of the user-item matrix. Which choice is the best is difficult to determine as it depends on the data, but KimRec seems to be a bit better.

Item-based is better than user-based. The reason for this is probably users rates fewer items compared to items that are rated and therefore the rating vectors are less sparse for item-based collaborative filtering.

9 Conclusion

This thesis demonstrated that implicit ratings can be extracted from the clickstream data of novomind and utilized as input for collaborative filtering. The results of this thesis indicate that the resulting recommendations are effective in providing users with potentially relevant items while also offering diversity within the scope of their interests. Additionally, this thesis showed that the recommendation process can incorporate a wide range of items and is able to create sufficient recommendations for each user.

The initial phase of this study involved the analysis and preprocessing of clickstream data. As a form of big data, clickstream data presented certain challenges that were addressed through the transformation of the data into a format conducive to fast-read operations and the correction of faulty data.

In the subsequent phase, two algorithms to extract implicit ratings from clickstream data, KimRec and HPCRec, were evaluated, modified and implemented. KimRec extracts users' and navigational behavior pattern on items from clickstream data while HPCRec generates implicit ratings based on item click sequences and purchases per session.

In the final phase, the extracted ratings were used as input for user-based and item-based collaborative filtering. The evaluation methods and chosen metrics were carefully analyzed. The algorithms KimRec and HPCRec were compared for their performance in user-based and item-based collaborative filtering. The results indicated that item-based is more effective than user-based as an item is more often rated than a user rates an item. Recommendations based on HPCRec are more accurate while those based on KimRec are more diverse and user-specific.

The challenges when it comes to working with big data were pointed out and ideas were presented to solve these challenges. The process of extracting implicit ratings from clickstream data and using these ratings to create recommendations through collaborative filtering was also demonstrated. The difficulties of evaluating the performance of collaborative filtering offline were also pointed out and relevant metrics were presented.

This thesis can serve as a valuable reference for future research on collaborative filtering based on clickstream data. Moreover, novomind can utilize this thesis to incorporate collaborative filtering into their system and enhance customer recommendations.

Clickstream data offers a wealth of information for generating recommendations in an e-commerce setting. There are different kinds of information which can be extracted from it such as the items a user views, adds to the basket or purchases, the time a user spends on the item pages or in which sequence a user clicked on items. This information can be used to create implicit ratings, eliminating the dependency on explicit ratings through a rating system. Collaborative filtering does not require specific knowledge about the items, user or domain, because it generates recommendations only based on ratings. In addition, collaborative filtering needs to be implemented only once. Only the extraction of the clickstream data must be adjusted from shop to shop.

9.1 Future work

9.1.1 Clickstream data structure

A major challenge of this thesis was the analysis and preprocessing of the clickstream data. The data is stored within a Hadoop cluster in a tabular structure and it is distributed over 18 tables. If all events for a user or a session should be extracted, then each table must be scanned for the UUID or session ID. This is an inefficient method due to the time it consumes. This thesis showed how beneficial a kind of metatable could be. The metatable could show where the events for each user and session are. This would require a unique identifier or pointer for each event row. Whether and how this is possible would still have to be investigated. However, an additional data warehouse for preprocessed clickstream data could be helpful. An approach on how to transform and persist the data in a PostgreSQL database for fast read operations is described in section 5.

Another idea is to restructure the clickstream data from a tabular structure into a directed graph structure and persist the data into a graph database. Each event and attribute, such as UUID, session ID, product ID or timestamp, could be a node. An edge from one event to another event describes the order of the events. An edge from an attribute to an event describes the attributes of an event. If all events for a user or a session ID should be extracted, then the events can be easily found over the attribute nodes. The advantages are fast reading operations for relational data and less redundant data as for each distinct attribute exists only one node.

9.1.2 Hadoop access

Access to the Hadoop cluster was difficult. The Hadoop cluster can only be accessed over an SSH connection. This makes it complicated to read data from an external device. A credential-protected API with read-only access would be very beneficial. If such an API exists, then the data can be extracted regularly, transformed and loaded into a data warehouse. From there the data can then be used for the recommendation process.

9.1.3 Online evaluation

The offline evaluation can indicate that the recommendation engine works well. However, only an online test with a live system and real users can provide clear results. An online evaluation requires integration into a live system. The integration must be well-tested so that possible errors, which could affect the live system, would be avoided. Also, the test strategy must be chosen. An A/B test seems to be the most suitable.

9.1.4 Reconstruct processes from multiple sessions

A session has a lifetime counter of 30 minutes which will reset after every event a user triggers. A new session is created if the 30 minutes are over and the user triggers another event. A user's full transaction process, including viewing the products, adding them to the basket and submitting the order, can be distributed over several sessions. It would be helpful to reconstruct one or more processes from a set of sessions, especially for the HPCRec algorithm.

Reconstruction of a user process is a difficult task as there is no guarantee, that a reconstructed process corresponds to reality. Indications like the time distances of the sessions or the *submitorder* and *logout* event (most likely the end of a process) can be used to reconstruct processes.

9.1.5 Finding the optimal period

In this thesis, a period of one month was selected for the evaluation. Further investigations are required to check if more or fewer data provide better results. If the period is too large, then the preprocessing may take too long. A small period will not provide enough ratings and underfit the user-item matrix. Presumably, a period between one and three months should provide good results.

9.1.6 Optimization of the recommendation process

This thesis showed that recommendations with memory-based collaborative filtering on ratings extracted from clickstream data provided by novomind are valuable. However, due to a time restriction, not all possible methods to optimize the recommendations could be tested. Therefore there are several options to improve the recommendations such as:

- Using a different similarity function such as *Pearson correlation coefficient* or *Spearman rank correlation.*
- Using a different preference function for user-based and item-based filtering.
- Using a model-based technique instead of a memory-based technique.
- Using a kind of "*expert system*" such as landmarks described in the paper *Speeding up Memory-based Collaborative Filtering with Landmarks* [LMZ17].
- Improving the extraction of the explicit ratings by using another approach to extract ratings or improving the methods used in this thesis (KimRec and HPCRec).

Bibliography

[Agg15] Charu C. Aggarwal. “Association Pattern Mining”. In: *Data Mining: The Textbook*. Cham: Springer International Publishing, 2015, pp. 93–133. ISBN: 978-3-319-14142-8. DOI: 10.1007/978-3-319-14142-8_4. URL: https://doi.org/10.1007/978-3-319-14142-8_4.

[Alp10] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, 2010.

[Ber18] James O. Berger. “Statistical Decision Theory”. In: *The New Palgrave Dictionary of Economics*. London: Palgrave Macmillan UK, 2018, pp. 12990–12996. ISBN: 978-1-349-95189-5. DOI: 10.1057/978-1-349-95189-5_1872. URL: https://doi.org/10.1057/978-1-349-95189-5_1872.

[Bre01] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[Bre84] Leo Breiman. *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[Doe08] John Doe. “Spearman Rank Correlation Coefficient”. In: *The Concise Encyclopedia of Statistics*. New York, NY: Springer New York, 2008, pp. 502–505. ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_379. URL: https://doi.org/10.1007/978-0-387-32833-1%5C_379.

[eMa22] eMarketer. *E-commerce as percentage of total retail sales worldwide from 2015 to 2021, with forecasts from 2022 to 2026*. July 2022. URL: https://www.statista.com/statistics/534123/e-commerce-share-of-retail-sales-worldwide/ (visited on 11/15/2022).

[FM16] Daniel F and Andreas M. *Big Data: Grundlagen, Systeme und Nutzungspotenziale*. Jan. 2016. ISBN: 978-3-658-11588-3. DOI: 10.1007/978-3-658-11589-0.

[Fou] Apache Software Foundation. *Impala*. Version 4.1.0. URL: https://impala.apache.org.

[Fou10] Apache Software Foundation. *Hadoop*. Version 0.20.2. Feb. 19, 2010. URL: https://hadoop.apache.org.

[FP18] Jonas Freiknecht and Stefan Papp. *Big Data in der Praxis*. 2018. DOI: 10.3139/9783446456013.fm.

[FPP07] David Freedman, Robert Pisani, and Roger Purves. "Statistics (international student edition)". In: *Pisani, R. Purves, 4th edn. WW Norton & Company, New York* (2007).

[Fri01] Jerome H Friedman. "Greedy function approximation: A gradient boosting machine". In: *Annals of statistics* 29.5 (2001), pp. 1189–1232.

[FS97] Yoav Freund and Robert Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: (1997), pp. 23–37.

[GK19] Garima Gupta and Rahul Katarya. *Recommendation Analysis on Item-based and User-Based Collaborative Filtering*. 2019, pp. 1–4. DOI: 10.1109/ICSSIT46314.2019.8987745.

[Gro] PostgreSQL Global Development Group. *PostgreSQL*. Version 14.4. URL: https://www.postgresql.org/.

[Her+04] Jonathan L. Herlocker et al. "Evaluating collaborative filtering recommender systems". In: *ACM Transactions on Information Systems* 22 (1 Jan. 2004), pp. 5–53. ISSN: 10468188. DOI: 10.1145/963770.963772.

[HHP21] Sherry He, Brett Hollenbeck, and Davide Proserpio. "The Market for Fake Reviews". In: *Marketing Science* (Aug. 2021). URL: https://mpra.ub.uni-muenchen.de/109381/.

[HL13] David W Hosmer and Stanley Lemeshow. *Applied logistic regression*. John Wiley & Sons, 2013.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "Elements of statistical learning". In: *Springer series in statistics* 27 (2009).

[IFO15] Folasade Isinkaye, Yetunde Folajimi, and Bolanle Ojokoh. "Recommendation systems: Principles, methods and evaluation". In: *Egyptian Informatics Journal* 16 (Aug. 2015). DOI: 10.1016/j.eij.2015.06.005.

[IS21] IDC and Statista. *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025 (in zettabytes) [Graph]*. June 2021. URL: https://www.statista.com/statistics/871513/worldwide-data-created/ (visited on 08/31/2022).

[Jal+18] Mahdi Jalili et al. "Evaluating Collaborative Filtering Recommender Algorithms: A Survey". In: *IEEE Access* 6 (2018), pp. 74003–74024. ISSN: 21693536. DOI: 10.1109/ACCESS.2018.2883742.

[Kim+05] Yong Soo Kim et al. "Development of a recommender system based on navigational and behavioral patterns of customers in e-commerce sites". In: *Expert Systems with Applications* 28 (2 Feb. 2005), pp. 381–393. ISSN: 09574174. DOI: 10.1016/j.eswa.2004.10.017.

[Klu+16] Thomas Kluyver et al. "Jupyter Notebooks – a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87–90.

[LMZ17] Gustavo R. Lima, Carlos E. Mello, and Geraldo Zimbrao. "Speeding up Memory-based Collaborative Filtering with Landmarks". In: (May 2017). URL: http://arxiv.org/abs/1705.07051.

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[RGS17] Mayank Rawat, Neha Goyal, and Soumya Singh. "Advancement of recommender system based on clickstream data using gradient boosting and random forest classifiers". In: *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2017, pp. 1–6. DOI: 10.1109/ICCCNT.2017.8204029.

[RM86] David E Rumelhart and James L McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Vol. 1. MIT press Cambridge, 1986.

[Sal92] Gerard Salton. *THE STATE OF RETRIEVAL SYSTEM EVALUATION*. 1992, pp. 44–449.

[SM95] Upendra Shardanand and Pattie Maes. "Social Information Filtering: Algorithms for Automating "Word of Mouth"". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '95. Denver, Colorado, USA: ACM Press/Addison-Wesley Publishing Co., 1995, pp. 210–217. ISBN: 0201847051. DOI: 10.1145/223904.223931. URL: https://doi.org/10.1145/223904.223931.

[VM03] Emmanouil Vozalis and Konstantinos G Margaritis. *Analysis of Recommender Systems' Algorithms*. 2003. URL: http://macedonia.uom.gr/.

[XE18] Ying Xiao and C. Ezeife. "E-Commerce Product Recommendation Using Historical Purchases and Clickstream Data: 20th International Conference, DaWaK 2018, Regensburg, Germany, September 3–6, 2018, Proceedings". In: Jan. 2018, pp. 70–82. ISBN: 978-3-319-98538-1. DOI: 10.1007/978-3-319-98539-8_6.

[Zie+05] Cai-Nicolas Ziegler et al. "Improving Recommendation Lists through Topic Diversification". In: WWW '05. Chiba, Japan: Association for Computing Machinery, 2005, pp. 22–32. ISBN: 1595930469. DOI: 10.1145/1060745.1060754. URL: https://doi.org/10.1145/1060745.1060754.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen.“

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: ______________________________

Vorname: ______________________________

dass ich die vorliegende Masterarbeit mit dem Thema:

Recommendation engine: Collaborative filtering based on customer behavior on an e-commerce website

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

________________	________________	______________________________
Ort	Datum	Unterschrift im Original