



Thomas Ulz, BSc.

Human Computation based Recommendation Technologies

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Alexander Felfernig

Institute of Software Technology

Graz, June 2016

Abstract

Recommender systems nowadays are deployed in nearly every branch of business with customers relying on the recommendation results to take decisions on possible purchases. Many systems are built for the respective use case and thus, do not provide the flexibility to scale to new product types or attributes. In addition to that, many solutions also do have other drawbacks. For instance, many recommender systems require users to create a user account and profile to be able to generate recommendations. Also, most systems do not present any information on the ranking process. Thus, possibly leaving people wondering about the recommendation quality and therefore losing confidence in the recommendation. Especially in the context of more complex items, a major drawback is the need for experts who are responsible for maintaining the information stored in the recommender's knowledge base. These people might become a bottleneck when it comes to adding new data and knowledge to the system.

In this master's thesis, a constraint-based recommendation approach is presented which eliminates the mentioned drawbacks. The recommender system is a modular, fully configurable system which allows people to collaborate in building the knowledge base for arbitrary recommender domains. The recommendation algorithm's decisions are based on information obtained through *human computation*. Results are easily explainable to users of the recommender. Rating items and receiving recommendations is done using *subjective* and *objective* facts about items, which users can specify as being their preferences. Thus, for getting recommendations, no user profile is needed.

In addition to the concept and recommendation algorithm proposed in this master's thesis, improvements which can be applied to the algorithms were introduced. Also, a web-service based server application and an HTML5 web interface were developed. To evaluate the implemented algorithms, a study was conducted where data from 356 participants was collected. Using this data, the algorithms presented in this master's thesis were compared against other baseline approaches which were clearly outperformed by those two methods.

Kurzfassung

Recommender Systeme werden heutzutage in nahezu jedem Geschäftszweig eingesetzt und von Kunden verwendet um mögliche Kaufentscheidungen zu treffen. Die meisten dieser Systeme wurden für einen speziellen Anwendungsfall entwickelt, weshalb sie meist nicht die Flexibilität besitzen um neue Produkttypen oder Attribute behandeln zu können. Zusätzlich haben die meisten Lösungen auch noch andere Nachteile. Zum Beispiel benötigen die Systeme oft einen Benutzeraccount und ein dazugehöriges Profil um Empfehlungen berechnen zu können. Die meisten Systeme zeigen außerdem keine Informationen bezüglich der Empfehlungsberechnung an. Das lässt viele Benutzer an der Qualität der Empfehlungen zweifeln, was auch das Vertrauen in die angezeigten Ergebnisse senkt. Ein gravierender Nachteil, speziell im Kontext von komplexeren Produkten, ist die Notwendigkeit von Experten, welche für das Pflegen der vorhandenen Informationen in der Wissensdatenbank verantwortlich sind. Im Hinblick auf das Hinzufügen von neuen Daten und Wissen stellen diese Experten einen potentiellen Engpass dar.

Aus diesen Gründen wird in dieser Masterarbeit ein Empfehlungsansatz basierend auf Beschränkungen (constraint-based) präsentiert, welcher die zuvor genannten Nachteile behebt. Das Recommender System ist modular und vollständig konfigurierbar, was Benutzern erlaubt bei der Erstellung der Wissensdatenbank für beliebige Produktgruppen zusammen zu arbeiten. Der Algorithmus zur Berechnung der Empfehlungen basiert auf diesen Informationen die mittels *Human Computation* gesammelt wurden. Empfehlungsergebnisse können den Benutzern des Recommenders verständlich erklärt werden. Das Bewerten von Produkten und das Anfordern von Empfehlungen basiert auf der Spezifikation von subjektiven und objektiven Eigenschaften der Produkte, welche von Benutzern als Anforderungen definiert werden können. Daher ist es nicht mehr notwendig ein Profil anzulegen, um Empfehlungen erhalten zu können.

Zusätzlich zum vorgestellten Konzept und Empfehlungsalgorithmus werden auch Erweiterungen für den Algorithmus gezeigt. Außerdem wurde eine web-service basierte Serveranwendung und eine dazugehöriger HTML5 Webseite entwickelt. Um die implementierten Algorithmen zu evaluieren wurde eine Studie durchgeführt in der Daten von 356 Teilnehmern gesammelt wurden. Anhand dieser Daten wurden die beiden in dieser Masterarbeit vorgestellten Ansätze mit anderen Basis-Algorithmen verglichen, welche von den beiden implementierten Ansätzen klar übertroffen wurden.

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date

Signature

Contents

1	Introduction	7
1.1	Motivation	7
1.2	Goals	9
1.3	Structure	10
2	Related Work	11
2.1	Recommender Systems	11
2.2	Collaborative Recommendation	13
2.3	Content-Based Recommendation	14
2.4	Knowledge-Based Recommendation	16
2.4.1	Constraint-Based Recommendation	16
2.4.2	Critiquing-Based Recommendation	17
2.5	Hybrid Approaches	18
2.6	Human Computation	19
2.7	Recommendation Algorithms	20
2.8	Beta Distribution Based Algorithms	21
3	Basic Recommendation Approach	23
3.1	Terminology	23
3.2	Averaging Support Values	31
3.3	Recommendation Approach	33
4	PEOPLEVIEWS	36
4.1	Architecture	36
4.1.1	PEOPLEVIEWS Server	37
4.1.2	Recommender as a Service	38
4.1.3	PEOPLEVIEWS Client	39
4.2	Used Software Components	39
4.3	User Interface	41
5	Enhanced Approaches	56
5.1	Beta Distribution Based Approach	56
5.1.1	The Beta Distribution	56

5.1.2	Calculating and Scaling Aggregated Support using Beta Distribution	58
5.2	User Preferences	65
5.2.1	Genetic Algorithm	65
5.2.2	Learning of User Preferences	66
5.3	Dataset Quality Assurance	67
5.4	Additional Features	69
5.4.1	Obfuscating	69
5.4.2	Recommendation Explanation	69
5.4.3	Similar Items	70
6	Evaluation	71
6.1	Dataset	71
6.1.1	Acquisition	72
6.1.2	Data	73
6.2	Baseline Methods and Evaluation Approach	73
6.2.1	Random	74
6.2.2	Most Frequent	74
6.2.3	Case-Based	74
6.2.4	Evaluation Approach	75
6.2.5	Evaluation Metric	75
6.3	Evaluation Results	76
6.3.1	Recommendation Approaches	76
6.3.2	User Preferences	80
6.3.3	Other Improvements	83
7	Limitations and Future Work	84
7.1	Limitations	84
7.2	Future Work	85
8	Conclusion	88

1

Introduction

The introduction to this master's thesis, given in this chapter, comprises a motivational part and goals of this work. In the motivation, reasons for using the selected recommender technique by listing drawbacks of other approaches are given. The goals then are defined such that a recommender system can be developed which does not inherit those mentioned drawbacks. Finally, the structure of the rest of this thesis will be explained.

1.1 Motivation

Recommender systems, although already very popular, are still becoming more and more used in various domains. Many of these recommenders work very well for the domain they were intended for, but often some drawbacks can be found as well. The recommendation approach presented in this master's thesis is trying to fix the problems that are going to be discussed in this section.

Today, most users know a recommender in the form of suggestions, where, when an item is viewed or even purchased, other *similar items* are suggested. These recommenders do use the information contained in item descriptions to find similar items. The corresponding recommendation approach, *content-based* filtering, will be explained in detail in Section 2.3. Also, by interacting with the system, users provide the recommender with additional data which also can be used to generate recommendations. Often, these items are shown as *items, other users have bought* to the potential buyer. This approach, called *collaborative-filtering*, will be discussed in Section 2.2.

Because of the simplicity of the rules used to generate recommendations, the previously mentioned types of recommenders easily can degenerate for a user. For instance, a very likely scenario might be multiple family members sharing a shopping account, as mentioned by Koen Verstrepen and Bart Goethals [VG15]. In this case, the parents might get recommendations for toys they bought for their child, without the chance to "repair" the recommender.

Also, nearly all systems that generate recommendations for users need the user to create an account. Thus, a user profile is generated which is often used by the recommendation approach for finding similar users. The items rated, viewed or liked by those similar users are then recommended to the other user.

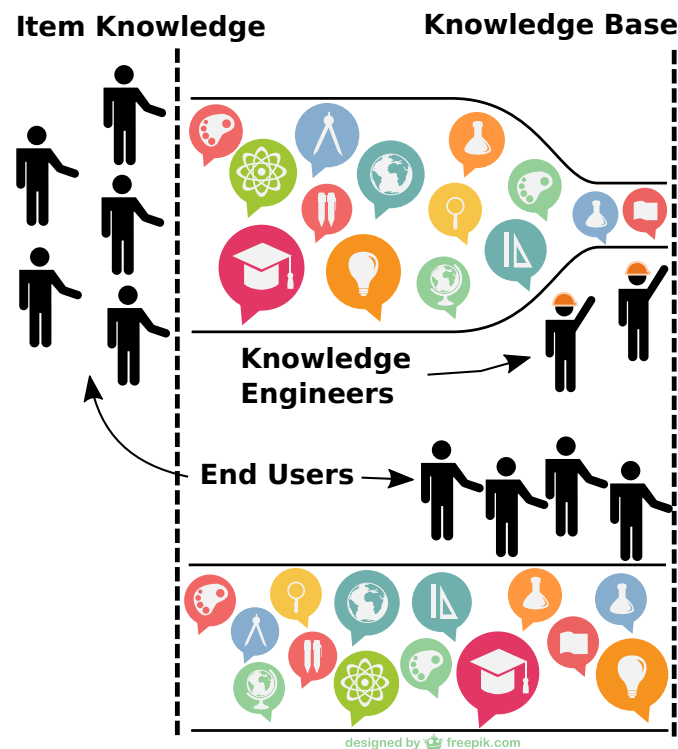


Figure 1.1: Knowledge Acquisition Bottleneck: Only a few knowledge engineers are contributing to a knowledge base. If end users with item knowledge are allowed to contribute as well, more information can be aggregated in less time.

By using item descriptions to generate recommendations, it is often unclear to a user why certain items are included in the recommendations. For example, when looking at the details of any *Terminator* movie, most recommender systems suggest titles like *Lincoln* or *Around the World in 80 Days*, because *Arnold Schwarzenegger* was in the cast of all of those movies. A user who is looking for action movies similar to *Terminator*, but is unaware that *Arnold Schwarzenegger* was in the cast of all of those films, might

doubt the quality of the recommender because no explanations are given.

Other types of recommenders rely on the ratings of users regarding the items contained in the system. However, in most of those approaches, an overall rating for the item is given, without the possibility to highlight very good or poor facts of the item in question. Thus, users need to find a compromise in the rating they are assigning an item. For instance, a user who might have bought a certain camera could be very satisfied with the excellent image quality. However, the camera might have very poor battery life. When rating the equipment with a simple star rating, the user has to decide what rating would be appropriate. Giving a high rating because of the image quality would ignore and not warn other users of the poor battery life, while a low rating because of the mentioned disadvantage would not highlight the excellent image quality. Also giving an in-between rating can be misleading because other users might think the overall quality of the camera is just not very satisfying.

Many types of recommenders also have in common that they mostly are designed for one item domain, e.g., books or movies. However, recommenders that are able to model any complex item domain and rely on explicit knowledge about the items also exist. For instance, a camera could be recommended based on the knowledge that it is suitable for sports photography. For these so called *knowledge-based* recommenders (see Section 2.4) it is necessary that the information used in the recommendation process is entered by people with knowledge about the recommender domain. We denote these people as *Knowledge Engineers*. In a system where either the information for multiple recommenders needs to be maintained or where new items need to be included on a regular basis, this might lead to long waiting times for this new information. We call this problem the *Knowledge Acquisition Bottleneck*, where a small group of *Knowledge Engineers* maintains the information contained in a *knowledge base*, although there would be a larger group of people that possess *Item Knowledge*. The problem of a *Knowledge Acquisition Bottleneck* is depicted in Figure 1.1.

1.2 Goals

Goals for this master's thesis were defined concerning the drawbacks of other recommendation techniques that were discussed in Section 1.1. The goals for the recommender approach presented in this work, therefore, are given in the following list.

Multi Domain. The recommender system and the algorithm to calculate recommendations should support multiple recommender domains. Thus, instead of specializing on a single recommender, it must be possible to define various domains such as *Canon DSLR Cameras*, *Cities* or *Skiing Regions*.

Descriptive Ratings. Ratings regarding items should not consist of a single value rep-

representing an overall rating for the respective item. Multiple criteria including *subjective* as well as *objective* facts about an item should be evaluable. All information must be used in the recommendation algorithm.

Many Contributors. To not run into a *Knowledge Acquisition Bottleneck*, any user must be able to contribute to the system's knowledge base. This includes tasks such as the specification of new recommender domains, adding and maintaining items as well as rating existing items. It should also be possible to get recommendations without the need to create an account first.

Recommendation Quality. The quality of recommendations generated for this configurable recommender should be equal or better than currently existing approaches for such systems. Also, generated recommendations should be explainable to the user to increase the confidence of users in the provided recommendations.

Inconsistency Management. If a user is specifying requirements which can not be matched by any item, no recommendation can be given. The system must be able to identify these inconsistencies and suggest repair actions to the user such that matching items can be recommended.

1.3 Structure

The remainder of this master's thesis is organized as follows. In Chapter 2 the terminology of recommender systems and different recommendation approaches are introduced. Related work for those definitions and algorithms is also given in that chapter. Chapter 3 introduces the concepts and terms that are necessary to build and describe the recommender system designed for this master's thesis. Also, a basic recommendation approach utilizing those concepts is presented in that chapter. PEOPLEVIEWS¹, the recommender system used to demonstrate and evaluate the concepts that were proposed in this work is presented in Chapter 4. There, the basic architecture, technical details, and the implemented user interface are discussed. Based on the basic approach shown in Chapter 3, a more sophisticated approach was developed. This method, as well as additional improvements that can be used for all two proposed recommendation algorithms, are discussed in Chapter 5. The recommendation approaches as well as the improvements were also evaluated as part of this master's thesis. The results can be seen in Chapter 6. Existing limitations of the proposed approaches as well as limitations of the implementation are listed in Chapter 7. This chapter also includes suggestions for future work. The master's thesis then is wrapped up in Chapter 8, where a conclusion about this work is given.

¹ PEOPLEVIEWS is a research project funded by the Austrian Research Promotion Agency under the Bridge-1 program.

2

Related Work

In this chapter, related work is used to explain basic terminology and concepts of recommender systems. Different categories of recommenders using various algorithms are discussed. Also, the idea of human computation is explained. Concluding, related work on recommender systems using Beta Distributions is listed.

2.1 Recommender Systems

The basic definition of a recommender system's goal is very similar in many publications. For instance, Prem Melville and Vikas Sindhwani give a definition in the *Encyclopedia of machine learning* [MS11].

"The goal of a Recommender System is to generate meaningful recommendations to a collection of users for items or products that might interest them."

However, by reading this definition, the question if such systems are necessary arises. Wouldn't most people be more confident in the recommendation for a good book or a restaurant, coming from a good friend? Do we need additional recommender systems? Paul Resnick and Hal R. Varian [RV97] argue that people often need to make choices without having sufficient personal experience in the respective topic. Recommendations from other people by word of mouth, recommendation letters, and movie or book reviews in newspapers might be used in everyday life. However, in the opinion of the authors, recommender systems can assist in and augment this natural selection process. The authors further define the function of a recommender system:

”In a typical recommender system people provide recommendations as inputs, which the system then aggregates and directs to appropriate recipients. In some cases the primary transformation is in the aggregation; in others the system’s value lies in its ability to make good matches between the recommenders and those seeking recommendations.”

Historically, the term recommender system was first coined in 1992 by Goldberg et al. [GNOT92] when the authors developed *Tapestry*. The reason for developing the system was to aid people with filtering mail. In their work, the authors realized that *content-based filtering* techniques did not provide satisfactory results when it comes to filtering unwanted mail. Therefore, they introduced the technique of *collaborative-filtering*. In the context of their work, collaborative filtering meant that people recorded their reactions to documents they read. By collaborating with other individuals who did the same, powerful filters could be created. The terms *collaborative-filtering* (see Section 2.2) and *content-based filtering* (see Section 2.3) are still relevant in nowadays recommender systems and thus, are discussed also in this chapter. Even earlier, Elaine Rich [Ric79] suggested in 1979, that computers might be able to model user behavior using stereotypes. By using those patterns, a system might be able to suggest books and novels which users might like. Also in this work, the author proposes the idea that the stereotypes should be learned by experience the system is gaining about users. The implemented system, Grundy, is evaluated, and it is shown that a good accuracy in predicting users preferences based on their categorization into stereotypes can be achieved.

Although included in many systems, recommenders mostly were unknowingly used by people when browsing the web. That changed in October 2006 when the *Netflix Prize* was announced. Netflix² released a dataset containing 100 million anonymous movie ratings and proclaimed a prize of 1 million USD for the first team to improve the recommendation accuracy of Netflix by at least 10%. The released dataset comprised the ratings of 480,000 randomly chosen users and nearly 18 thousand movie titles. At that time, no dataset of that size was publicly available. Bennett and Lanning [BL07] analyzed the dataset and stated that the permission to use the dataset for other non-commercial research purposes will boost work in that field. The announcement of the Netflix Prize brought the idea of recommender systems to a wider audience and triggered many related publications. Bell and Koren [BK07] for instance give an overview of the findings and improvements made by the winning team of the Netflix Prize. The contributions include a new method that takes interactions between neighbors into account when calculating nearest neighbors, which can be used in many machine learning applications.

Traditionally, recommender systems use algorithms to suggest and rank items that can be categorized into three different categories, as was done for instance by Gediminas Adomavicius and Alexander Tuzhilin [AT05]:

² <https://www.netflix.com/>

- Collaborative Recommendation
- Content-based Recommendation
- Hybrid Approaches

Felfernig et al. [FJN⁺14] extend this categorization by *Knowledge-Based Recommendation*, which is the category of recommender system used for this master's thesis. The following sections are going to introduce and discuss the above mentioned different recommendation principles.

2.2 Collaborative Recommendation

As with the general definition of recommender systems, there are also many similar definitions for collaborative filtering approaches. One is given by Schafer et al. [SFHS]:

”Collaborative Filtering is the process of filtering or evaluating items using the opinions of other people.”

The authors also make the case that the concept of collaborative filtering has been around for centuries - humans sharing opinions with others. However, as the authors in that paper stated, by using computers and the Internet, the concept of collaborative filtering can advance beyond the basic word of mouth. Instead of using only the opinions of a hand full of people, the Internet allows to consider the assessments of thousands or millions of people.

The core concepts of a collaborative filtering approach are *Users*, *Items* and *Ratings*, as noted by Ekstrand et al. [ERK11]. In this work, the authors state that the information domain for a collaborative filtering system consists of users who specify their preferences for various items. The preference specified by the user for a certain item is then denoted as a rating. The authors also state that the ratings can be of varying form, depending on the implementation. For instance, scales of 0-5 stars or binary scales (like/dislike) can be used. In any case, all (User, Item, Rating) triples can be represented in a matrix where all users are assigned in one dimension of the matrix, while all items are assigned in the other dimension. The ratings are then stored in the respective cells of that matrix. The authors also further distinguish between *User-Based* and *Item-Based* collaborative filtering in their work.

The first *User-Based* collaborative filtering approach was introduced by Resnick et al. [RIS⁺94] in 1994. The intention of their proposed system, GroupLens, was to help people find articles they like in Usenet. After reading an article, users were presented with the possibility to rate the article. The ratings then were aggregated on rating servers, called *Better Bit Bureaus*. Those rating servers then suggested articles to users based on

the assumption, that users who agreed on ratings in the past, will probably agree again.

In contrast, *Item-Based* collaborative filtering approaches rely on the similarity between items, as Sarwar et al. [SKKR01] stated in their work:

”Item-based techniques first analyze the user-item matrix to identify relationships between different items, and then use these relationships to indirectly compute recommendations for users”

To measure how similar a pair of items is to each other, different techniques are discussed in that paper, including item-item correlation or cosine similarities between item vectors. In that approach, item vectors which describe the *features* of an entity are stored in an n-dimensional vector, where n is the number of features taken into account. The cosine similarity metric then measures the angle between two item vectors to determine their similarity.

Herlocker et al. [HKTR04] give a thorough analysis and comparison of different collaborative filtering approaches. The authors define tasks and evaluation metrics, as well as datasets to evaluate collaborative filtering algorithms, such that reproducible evaluations can be done. They state, that most algorithms work best on a specific problem (for instance far more items than users) and thus, algorithm performances do not scale to other datasets. Also, many different evaluation metrics are applied in literature, thus, no fair comparison can be done.

Finally, the application of collaborative filtering based recommender systems requires to answer the following questions [JZFF10]:

- How can users similar to the user who is looking for recommendations be found?
- How is similarity measured?
- How to handle new users, for whom no information is available yet?
- How to deal with new items, for which no ratings exist yet?
- What if only a few ratings exist, that can be used to generate recommendations?

2.3 Content-Based Recommendation

In their research paper, Beel et al. [BGLB15] did a literature survey of more than 200 research articles, coming to the conclusion that in 55% of all recommendation approaches content-based filtering is used, while only 18% of the reviewed approaches applied

collaborative filtering (the authors mainly discussed content-based and collaborative filtering, therefore no details are given about the remaining 27%).

Content-Based Recommendation can be defined in one single sentence, as for instance Michael J. Pazzani and Daniel Billsus [PB07b] wrote:

“Content-based recommendation systems analyze item descriptions to identify items that are of particular interest to the user.”

As the authors also noted in their work, the details available for items differs in recommender systems. Therefore, the methods used for analyzing item information are manifold. One of the most popular methods used in such systems is *tf-idf* which is a shortcut for *term frequency-inverse document frequency*. The method is discussed in many publications, for instance in *Mining of massive datasets* by Rajaraman et al. [RUUU12] where the authors define *tf-idf* as a numerical statistic, which can be used to reflect the importance of a certain word in a document relative to a collection of documents or a corpus. After extracting the top *n* most relevant words describing an item, the authors suggest to use for instance *Jaccard distance* or *cosine distance* measures to calculate which items might be interesting to users. The user vector utilized in those comparisons according to the authors could, for instance, be constructed using a history of entities a user already interacted with, such as items which details where viewed.

Most papers and articles state that content-based filtering also can be reduced to a binary text classification problem. Items are recommended to users based on their descriptions, who then either like and agree with the recommendation or dislike the recommended item. An early work by Lewis et al. [LSCP96] elaborates on training algorithms for linear text classifiers. The work introduces two algorithms, *Widrow-Hoff* and *EG* which can be used for this tasks.

The application of content-based recommendation systems requires to answer the following questions [JZFF10]:

- How are systems able to continuously improve user profiles by automatically acquiring user information?
- How can the similarity of items to user profiles be calculated?
- What methods could be applied to extract the information contained in item descriptions?

2.4 Knowledge-Based Recommendation

A third recommendation technique, knowledge-based recommendation was described by Robin Burke [Bur00] as follows:

"A third type of recommender system is one that uses knowledge about users and products to pursue a knowledge-based approach to generating a recommendation, reasoning about what products meet the user's requirements."

As Felfernig et al. [FFJZ06] have stated, by utilizing deep knowledge about the item domain when generating recommendations, several advantages of a knowledge-based approach make it possible to:

"[...] (b) to explain solutions to a customer, and (c) to support customers in situations in which no solution can be found."

Many approaches, including the one proposed by Robin Burke [Bur99] combine collaborative aspects with features of knowledge-based recommenders. One drawback of collaborative filtering approaches is the so-called *cold start problem*, where no recommendations can be made for users who newly registered with the system and thus, no information about the user is available. In his approach, the author suggested to use a knowledge-based approach while the amount of information is small, and to use collaborative filtering only as a post-filter for the knowledge-based approach.

The questions that have to be answered when applying knowledge-based recommenders are [JZFF10]:

- What types of knowledge regarding the item domain can be represented in the knowledge base?
- What techniques can be used for item selection and the ranking of the resulting item set concerning user requirements?
- If no history of a user exists yet, how can a user profile be acquired? Also, how are user's explicit preferences taken into account?

2.4.1 Constraint-Based Recommendation

A particular form of knowledge-based recommenders are so-called *constraint-based* recommenders, where according to Alexander Felfernig and Robin Burke [FB08], the recommendation process can be viewed as a process of constraint satisfaction. In their definition, some constraints are coming from the user, while others are derived from the product domain. Products that can satisfy those constraints are then viewed as good

recommendations.

In *Developing Constraint-based Recommenders* by Felfernig et al. [FFJZ11] the authors define the constraint satisfaction problem constructed in constraint-based recommender systems. Also in this work, the entities necessary to build the *knowledge base* of such a constraint-based recommender are defined. In this master's thesis, those definitions were used as a starting point when developing the recommendation algorithm. The concepts were extended in this work and are discussed in Section 3.1.

While collaborative and content-based filtering techniques are mostly applied to simple item domains such as books, movies or restaurants, constraint-based recommenders have been implemented in a wide range of different fields. For instance, Jannach et al. [JZF09] and also Zanker et al. [ZFH⁺08] discussed the usage of constraint-based recommenders in tourism. Felfernig et al. [FJS⁺15] elaborate on the possibility to use constraint-based recommenders in the context of financial services. There, as also noted by Felfernig et al. [FFJZ06], recommenders need to generate recommendations which adhere to legal regulations.

More recently, Felfernig et al. [FHN⁺14] introduced the idea of a multi-domain constraint-based recommender which can be configured to support arbitrary item domains. Also in this paper, a study was done showing that users although willing to use and contribute to a constraint-based recommender's knowledge base, they are only prepared to invest a short period of time. Felfernig et al. [FUH⁺15] also published a paper discussing the basic recommendation approach for such a multi-domain recommender system. That paper includes parts of this work, as it was written in the context of this master's thesis.

2.4.2 Critiquing-Based Recommendation

A second special case of knowledge-based recommendation is *critique-based* recommendation. The concept of critiquing was explained, for instance, by Robin Burke [Bur00]. The main idea is that users can specify requirements the currently displayed item is not satisfying. In other words, they are criticizing the current recommendation. For instance, for a currently displayed digital camera, a user might state that the highest possible ISO value is too low and therefore requests a new recommendation.

Li Chen and Pearl Pu [CP12] state that in a critiquing-based recommender system, users do interact with the system in a kind of conversational style. In contrast to that, users get recommendations in a single interaction in other approaches. The authors state that through the use of critiquing feedback systems can easily learn users' profiles and therefore are able to make better recommendations in future sessions.

Dynamic critiquing, a particular form of critiquing, is explained by Reilly et al.

[RMMS04]. The authors point out that critiquing, although applicable to a wide range of domains, is limited to a single feature such as price or camera resolution. Therefore, they propose to extend critiquing to allow compound critiques, which allows critiques over multiple features. The authors further state that by using this generalized approach, explanations for users become possible. One example of a compound explanation would be "30% of the remaining cameras have a higher maximum ISO and higher resolution".

One example application of critiquing-based recommenders are mobile recommenders, as Francesco Ricci and Quang Nhat Nguyen [RN05] state in their work. They argue that the conversational style recommendation process might be preferred by on-the-move travelers. In their work, they show that giving critique-based feedback is suited for the comparatively small (smart-)phone screens because only 2-3 clicks are necessary to provide feedback to the system.

2.5 Hybrid Approaches

An overview of the recommendation techniques mentioned above is given by Burke et al. [BFG11] where the differences between the approaches are highlighted. Also Jannach et al. [JZFF10] note that both concepts (content-based and collaborative) have advantages. Collaborative filtering approaches do not need explicit information regarding the items, such as descriptions, as user ratings are used to recommend items. However, to guarantee meaningful recommendation results, a significant number of users is required. In contrast to that, content-based approaches do not need larger user groups to provide reasonable recommendation accuracy. Because both concepts have their advantages, much research was put into hybrid approaches that combine collaborative with content-based filtering.

Fab, proposed by Marko Balabanović and Yoav Shoham [BS97] is a Web recommendation system introduced in 1994. The goal of this work was to recommend web pages to users, much like search engines do nowadays. The authors state that the goal of their work was to combine the advantages of collaborative and content-based filtering while inheriting the disadvantages of neither approach. In their approach, user profiles are used to reflect interests. Then similar user profiles are used for collaborative filtering to increase the recommendation accuracy.

In the work by Basu et al. [BHC⁺98], the authors denote collaborative filtering as social filtering. They claim that by combining social information with item information in a movie recommender, better recommendation results can be achieved. In their approach, the authors do not rank the recommended items but just try to predict if users will like or dislike a suggested movie. To achieve this, users were grouped into users who liked, for instance, a certain genre, e.g., *drama*. For this subset of users, traditional collaborative filtering could then be applied.

In addition to collaborative and content-based filtering, Pazzani et al. [Paz99] also discuss *demographic-based* recommendations. There, personal information of users such as age, gender and education is used to generate restaurant recommendations. In their approach labeled *Collaboration via Content*, content-based user profiles are used to find similar users for which then weighted collaborative filtering techniques are applied. The weights used are results of the applied term extraction method used to generate the user profiles.

Robin Burke [Bur02] gives an overview of various other hybrid approaches that combine different recommendation techniques. One of the methods introduced is a hybrid approach that combines collaborative filtering with critique-based feedback to recommend restaurants. This approach can significantly increase recommendation quality.

2.6 Human Computation

A definition of human computation is given by Edith Law and Luis von Ahn [LVA09]:

“Human computation is the idea of using human effort to perform tasks that computers cannot yet perform, usually in an enjoyable manner.”

Human computation, however, is often confused with crowdsourcing, as Alexander J. Quinn and Benjamin B. Bederson [QB11] note in their paper. They describe the differences as follows:

“Whereas human computation replaces computers with humans, crowdsourcing replaces traditional human workers with members of the public”

Two key characteristics of human computation, are given:

- The problems solved by human computation generally fit the paradigm of computation, and therefore those problems might be solvable by computers someday.
- Human participants are guided by a computer systems user interface or process.

Amazon Mechanical Turk³ (MTurk) provides an easy possibility to crowdsource tasks, which then can be used for human computation. Because of its simplicity, MTurk is utilized in many applications. Little et al. [LCGM09], [LCGM10] presented TurKit, a framework to crowdsource tasks to MTurk. The framework allows for parallel computation of tasks by humans, but also for iterative tasks. By enabling iterative tasks, workers can build upon the work done by others. Possible use cases that were discussed

³ <https://www.mturk.com/>

included iterative text improvements or handwriting recognition.

The concept of human computation is also applicable to recommender systems. In a study done by Krishnan et al. [KNN⁺08], the recommendation performance of an *impersonal* recommender algorithm compared to the *manual* recommendations done by power users of a movie database was evaluated. The authors found out that although the algorithm on average outperformed the human recommendations, in many cases the recommendation accuracy of humans was higher than the algorithm's.

However, most of the time human computation in recommender systems is used to collect information that can be used to generate recommendations. As mentioned previously, human computation tasks often can be done in an enjoyable fashion. Therefore, most approaches use games (with a purpose) to collect information from users. For example, Greg Walsh and Jennifer Golbeck [WG10] introduced Curator, a game used for collection recommendation. Collection recommender systems suggest items that work together very well as a group (for instance, steak and mashed potatoes match well as a group, while chocolate and mashed potatoes would not match). In another approach, Dugan et al. [DMM⁺07] introduced *Dogear*, a game which helps to collect data for bookmark and website recommendations.

2.7 Recommendation Algorithms

There is a broad range of recommendation algorithms available for the different types of recommender systems that were discussed in this chapter. Also, as noted by Herlocker et al. [HKTR04], most papers use adapted datasets or evaluation metrics. Therefore, no fair comparison between all available algorithms was performed to this date.

The most widely used algorithm for collaborative filtering recommender systems is *matrix factorization* as discussed for instance by Koren et al. [KBV09]. In this approach, the user-item matrix is decomposed into several matrices, which can be used to construct the original matrix again. One such method, for instance, is the SVD-decomposition that is used to disclose the *latent factors* hidden in the user-item matrix.

Other very popular approaches using the user-item matrix to disclose latent factors are based on *Latent Dirichlet Allocation (LDA)*, first introduced by Blei et al. [BNJ03]. Technically, LDA is a probabilistic model for collections of data, for instance, text corpora. The authors describe LDA as follows:

“LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities.”

Blei et al. [BNJ03] also show that LDA is applicable for collaborative filtering approaches. Deepak Agarwal and Bee-Chung Chen [AC10] use latent dirichlet allocation for matrix factorization. The authors claim their approach works best on recommender systems, where a representation of items with bags-of-words is possible. This fact applies, for instance, in web search or ad targeting as the authors noted in their work.

Content-based recommendation algorithms use a broad range of different similarity measures to determine similar items. Spertus et al. [SSB05] give an overview of possible algorithms and evaluate them based on their performance on a large scale dataset.

The constraint-based recommender introduced in this work builds on the work done by Felfernig et al. [FUH⁺15]. Also, to adapt the approach discussed in that paper, a second alternative approach is introduced in this work, utilizing *Beta Distributions* to model the knowledge base.

2.8 Beta Distribution Based Algorithms

The Beta Distribution is a probability distribution used to model the behavior of random variables and can be used in many disciplines, as Arjun Gupta and Saraless Nadarajah [GN04] noted.

For instance, Audun Jsang and Roslan Ismail [JI02] use Beta Distributions as a reputation system. In their paper, reputation denotes the aggregation of user ratings. Each rating is modeled as an event in the Beta Distribution with a prior probability of happening and thus has an influence on the shape parameters α and β which define the form of the Beta Distribution's probability density function. The authors also introduce the idea of adaptive weighting and forgetting in their approach, which they show is superior to other more basic aggregation methods. A similar approach was demonstrated by Ahmad Abdel-Hafez and Yue Xu [AHX15], who use statistical methods to calculate reputation models which are utilized to generate recommendations. Reputations are used to report on the quality of items on the web and are aggregated user ratings, such as star ratings. Reputation according to the authors should include the rating itself as well as the time of the rating, trust between users and the reputation of the rating user. If incorporated correctly into a recommender system, the authors claim that more accurate recommendations are possible. A Beta Distribution based reputation model is used amongst others, providing the best recommendation accuracy in the author's evaluation.

Yin et al. [YSC⁺13] use the Beta Distribution as part of a recommendation algorithm specifically developed for a location-content-aware recommender with sparse user-item matrix data. The approach presented in that paper, LCARS, is used to recommend venues and events in cities. The algorithm proposed consists of a modeling part which utilizes

LDA, where Beta Distributions are used to model priors for generating user models. The authors claim that their approach can deal with sparse data such as users looking for recommendations when visiting a new city. Using classical collaborative filter algorithms, no satisfactory recommendation would be possible with that constellation.

Chung et al. [CHH13] propose βP , an algorithm to counteract attacks on collaborative recommender systems by using Beta Distribution to detect malicious user profiles. So-called *shilling attacks* try to introduce attacking profiles which then try to manipulate the user-item matrix by providing biased (either very high or very low) ratings for certain items. The authors use Beta Distributions to model user ratings and thus can precisely detect malicious ratings and the associated user profiles.

Condliff et al. [CLMP99] propose a recommender framework that incorporates user ratings, user features and item features in one single framework. They claim that their approach can deal with the so-called cold-start problem as well as with sparse user-ratings. In their algorithm, a Bayesian approach is used where Beta Distributions model a prior probability for predictions regarding the recommended items. The authors of the paper, however, note that evaluations have shown mixed results. For some datasets, excellent recommendation accuracy was achieved while for other data, poor performance resulted.

3

Basic Recommendation Approach

In this chapter, a basic recommendation approach for a constraint-based recommender system is discussed. The approach described here is based on the approach described by Felfernig et al. in [FUH⁺15].

Before the recommendation algorithm is discussed, the necessary terminology to describe the approach is defined. The terminology also includes concepts used in the practical implementation of the proposed recommendation algorithm. For all introduced concepts, examples are given. The recommendation approach can be split into two parts. The first part can be interchanged with a different methodology and is discussed in Section 3.2, while the second part which is introduced in Section 3.3 will also be used for the alternative algorithm introduced in Section 5.1.

3.1 Terminology

There are a couple of basic entities that need to exist in every recommender system, namely *items*, *users* and a *recommender*. In the context of a recommender system, the relationship between those entities is fairly clear. Users interact with a recommender to get a set of suggested items. For a *constraint-based recommender*, those entities need to be supplemented by some constructs which are used to build the *knowledge base* against which *constraints* can be evaluated. In the chapter *Constraint-based recommender systems* in *Recommender Systems Handbook* [FFJZ15], Felfernig et al. give an overview of what is needed to build a constraint-based recommender system. We further expand the entities listed there, to enable a constraint-based recommender system which can be extended with new recommenders by users. The following list gives an overview of the

entities used in PEOPLEVIEWS. Also, an exemplary recommender that was employed in the evaluation of this master's thesis will be defined.

User $u \in U$. It is possible to get recommendations from PEOPLEVIEWS as an anonymous user. For all other interactions with the system, a user u needs to register and thus create a user account in our system. The information required to create such an account are a username, email, and password. Also, a profile picture can be uploaded. Registered users are able to contribute fully to the knowledge base of PEOPLEVIEWS by creating new recommenders, adding items to existing recommenders or evaluating existing items. Table 3.1 shows an exemplary list of users.

	Username	Email	Password
u_1	Sally	sally@email.com	*****
u_2	Mike	mike@email.com	*****
u_3	Mary	mary@email.com	*****
u_4	Susan	susan@email.com	*****
u_5	Bob	bob@email.com	*****
u_6	Eve	eve@email.com	*****

Table 3.1: An exemplary list of users. Additional information such as profile picture was omitted in this table.

Item Attribute $ia \in IA$. To describe *objective* properties or *hard facts* about items, *item attributes* are used. The attributes can be either a number, text or an enumeration of possible values. Item attributes should be evident to each user, or they should be easy to look up. Because of this fact, item attributes need to be specified by a user when adding new items to PEOPLEVIEWS. To compare and rank items based on item attributes, an order relation needs to be specified. For numeric attributes, one of the following similarity measures [McS03] has to be chosen: *more is better (MIB)*, *less is better (LIB)*, *near is better (NIB)* or *equal is better (EIB)*. Text and enumeration attributes can only be compared using the *EIB* metric. Those order relations specify whether a certain value is better if higher, lower or nearer to the compared to value. By defining a similarity measure for item attributes, it is possible to use them in the recommender algorithm when ranking items. If an item attribute should only be used as additional information for items but not to filter and rank items, it is possible to exclude the attribute from being used in the recommender. For instance, the producer of an item could be relevant information to display which might not be a necessary attribute for getting recommendations. Possible examples of item attributes are the maximum resolution of a digital camera or the population of a city. All item attributes used in the example recommender are listed in Table 3.2.

Item $i \in I$. The entities recommended in our system are called *items*. All registered users are allowed to create new items. Upon creation, information such as the

	Attribute	Type	Similarity Measure	Question to User creating Item	Filter
ia_1	Megapixel	Number	MIB	What is the maximum resolution of this camera in megapixel?	T
ia_2	Max ISO	Number	MIB	What is the maximum ISO value this camera can support?	T
ia_3	Price	Number	LIB	What is the suggested retail price of this camera?	T
ia_4	Sensor Format	{full, APS, APS-C, APS-H}	EIB	What is the sensor format of this camera?	T

Table 3.2: Item attributes used for a Canon DSLR recommender.

item's name, description, tags and image need to be specified. In addition, all *item attributes* become mandatory inputs for new items and thus, need to be answered by the user creating a new item. Table 3.3 lists example items as used in our Canon DSLR recommender. Because of space limitations, only the name and item attribute values are shown.

	Name	Desc.	Tags	Megapixel	Max ISO	Price	Sensor Format
i_1	EOS 7D	[...]	[...]	18.0	12800	1699	APS-C
i_2	EOS 550D	[...]	[...]	18.0	6400	350	APS-C
i_3	EOS 5D Mark III	[...]	[...]	22.3	25600	2800	full
i_4	EOS 70D	[...]	[...]	20.2	12800	900	APS-C
i_5	EOS 1D X	[...]	[...]	18.1	51200	3000	full
i_6	EOS 5DSR	[...]	[...]	53.0	102400	3600	full
i_7	EOS Rebel T6i	[...]	[...]	24.7	25600	750	APS-C
i_8	EOS 1200D	[...]	[...]	18.1	6400	399	APS-C
i_9	EOS 7D Mark II	[...]	[...]	20.2	16000	1799	APS-C
i_{10}	EOS 80D	[...]	[...]	24.2	16000	1199	APS-C

Table 3.3: An example list of items used in a Canon DSLR recommender. Information such as description, tags and item image were omitted in this table because of space limitations.

User Attribute $ua \in UA$. In addition to *item attributes* or hard facts, we also define

user attributes. These attributes are needed to describe *soft* or *subjective* facts about items. Thus, the perception of different users regarding the same *user attribute* might be completely different. For instance, a user attribute for our Canon DSLR recommender might be *Field of Application* which might describe for which use case a given camera is best suited for. User u_1 considers the item i_1 (Canon EOS 7D) to be best suited for sports photography, while user u_2 thinks it can be best used to capture macro images. As *user attributes* are subjective facts about items, every user $u \in U$ can *assign* user attributes to items. However, because every user might have a different opinion regarding a *user attribute*, it is necessary to specify a set of possible **User Attribute Values** ($uav \in \text{dom}(\text{ua})$) to limit the assigned attributes to a predefined set. Allowing every user to enter values freely would result in no usable data. For instance, for the previously mentioned user attribute *Field of Application*, the *user attribute values* might be: *Macro*, *Portrait*, *Sport*, *Tele* and *Landscape*. Along with those user attribute values, also an attribute name has to be specified. Because users can assign user attribute values, it is also necessary to specify if for a given user attribute only choosing one user attribute value (Single Choice) or choosing multiple values (Multiple Choice) is allowed. Finally, to help users understand the attributes, a question text which will be shown to users needs to be defined as well. Users can assign user attributes to items by explicitly choosing an item to evaluate. There, they are asked to assign one or more user attribute values to the selected item. The user attributes with their corresponding user attribute values for the Canon DSLR recommender used as an example recommender in this master's thesis are shown in Table 3.4.

Recommender $r \in R$. Using the above definitions of *item attributes*, *user attributes* and *user attribute values*, it is possible for users to specify new recommenders. When creating a new recommender, information such as name, a description, tags and a recommender image are needed. In addition, the creator of a recommender needs to define all item attributes and user attributes (with the corresponding sets of user attribute values) belonging to the new recommender. As soon as a new recommender is created, it is possible to add items to it. A recommender, therefore, contains item attributes, user attributes with user attribute values and added items. The *Canon DSLR* recommender used throughout this master's thesis is shown in Table 3.5.

Support s . As mentioned, users are able to assign *user attribute values* to *items* to describe their subjective facts. Assigning user attribute values to an item, however, might not fully express the users opinion about that item. If we, for example, consider the same Canon DSLR recommender as before, user u_1 assigned the user attribute value *Sport* to item i_1 regarding the user attribute *Field of Application*. Without specifying any further information, this designation would state that the user thinks the given item fully (100%) supports the chosen user attribute value. Therefore, *support* s is introduced, which needs to be specified by the user for every *item - user attribute value* pair that is specified by her. Thus, we do not force

	Attribute	Question to User evaluating Item	SC / MC	User Attribute Values (dom(ua_i))
ua_1	Field of Application	For which field of application is this camera suited for?	SC	{Macro (uav_{11}), Portrait (uav_{12}), Sport (uav_{13}), Tele (uav_{14}), Landscape (uav_{15})}
ua_2	Experience Level	What is the suggested level of experience a user should have using this camera?	MC	{Absolut Beginner (uav_{21}), Amateur (uav_{22}), Expert (uav_{23})}
ua_3	Durability	What level of durability would you expect from this camera?	SC	{Bad (uav_{31}), Moderate (uav_{32}), Good (uav_{33})}
ua_4	Value for Money	What do you think is the value you get for your money when buying this camera?	SC	{Good Deal (uav_{41}), Price is OK (uav_{42}), Too Expensive (uav_{43})}
ua_5	Loss of Value	How fast do you think will this camera loose its value if bought new?	SC	{Stable Value (uav_{51}), Loses in Value Slowly (uav_{52}), Loses in Value Fast (uav_{53})}

Table 3.4: User Attributes and corresponding User Attribute Values for a Canon DSLR recommender.

	Name	Desc.	Tags	Owners	Item Attributes	User Attributes	Items
r_1	Canon DSLR	[...]	[...]	$\{u_1\}$	$\{ia_1, ia_2, ia_3, ia_4\}$	$\{ua_1, ua_2, ua_3, ua_4, ua_5\}$	$\{i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8, i_9, i_{10}\}$

Table 3.5: Canon DSLR recommender containing all defined item attributes and user attributes. Information such as description, tags and image are omitted because of space limitations. Also, all items added to the recommender are listed.

users into yes or no decisions but rather let them specify how well they think a given item supports the chosen user attribute value. Because *support* is expressed as a percentage, it ranges from 0% (item does not support user attribute value at all - same as not assigning it at all) to 100% (item fully supports the assigned user attribute value). As a result, two users might think the same item *supports* the same user attribute value, but they may assign different supports. For example, user u_1 could specify that item i_1 supports the user attribute value *Sport* with 90%, while user u_2 assigns the same item a support of 80% for the user attribute value *Sport*. A

complete example of users and their assigned user attribute values including support can be seen in Table 3.6.

Evaluation e . If a user assigns user attribute values and supports for various user attributes, we denote this action as *evaluating* the item. Therefore, one row in Table 3.6 is referred to as the evaluation of a certain item by the respective user.

Microtask mt . Evaluating an item with respect to all specified user attributes in a recommender might take some time for users. For instance, if a user wants to specify supports for all user attributes listed in Table 3.4, at least five support values need to be entered. Because of this fact, we introduce microtasks, which are simple tasks that are assigned automatically to users. A microtask always refers to one item and user attribute. Thus, users can evaluate items but do not need to invest much time. To assign microtasks to users such that the probability of users solving the microtasks is high, an algorithm discussed in a different master's thesis is developed. That algorithm tries to consider many aspects of the system. For users, their experience in the respective recommender domain and the corresponding expertise in that field are derived from previous interactions with the system. Also, the potential interest of a user in solving the microtask is estimated by considering the users current workload and her previous contributions to the domain in question. Finally, also the importance of a microtask is considered. Using all of those mentioned aspects, the required number of best matching users will be found. In addition to the microtasks that ask users to specify support for a given item - user attribute value combination, also CAPTCHA microtasks are introduced. These tasks should be easily solvable for human users and therefore can be used to ensure data quality (see Section 5.3).

Aggregated Support $support_{\Sigma}$. As every user can specify *supports* for *item - user attribute value* pairs, we need to aggregate the given *supports*. Aggregation is done for each *item - user attribute value* pair respectively, accumulating the *supports* of all users who assigned a *user attribute value* and the corresponding *support* for that pair. Thus, we can provide aggregated supports which then describe items with respect to the opinion of all users giving feedback for the item. In this master's thesis, a basic approach on how to calculate the aggregated support (Section 3.2) as well as an enhanced approach using Beta Distribution (Section 5.1) are given. The approaches including examples are discussed in their respective sections of this work.

Game g . As another alternative form to collect user inputs, a game is introduced. In this game, users get ten different questions regarding *item - user attribute value* combinations. For those ten combinations, users should guess the current aggregated support. By doing so, users implicitly specify supports for the given *item - user attribute value* combinations. Also, the implementation of the game was done as part of a different master's thesis and therefore is not further discussed in this work.

U	I	Field of Application	Experience Level	Durability	Value for Money	Loss of Value
u_1	i_1	Sport(1.00)	Exp(0.90)	Mod(1.00)	Exp(0.95)	LiVS(0.85)
u_2	i_1	Sport(0.90)	Exp(0.95)	Good(0.95)	Pr OK(1.00)	LiVS(0.95)
u_3	i_1	Sport(0.90)	Am(0.85),Exp(0.90)	Good(0.90)	Exp(0.90)	SV(0.95)
u_4	i_1	Sport(0.90)	Exp(0.90)	Good(1.00)	Pr OK(0.90)	SV(1.00)
u_5	i_2	Sport(0.95)	AB(0.95)	Mod(0.90)	GD(1.00)	LiVF(0.90)
u_1	i_3	Macro(1.00)	Am(0.60),Exp(0.90)	Good(0.90)	Exp(0.70)	SV(0.85)
u_2	i_3	Portr(0.80)	Exp(1.00)	Good(0.95)	Pr OK(0.95)	SV(0.70)
u_3	i_3	Portr(0.80)	Exp(0.95)	Good(0.90)	Pr OK(0.95)	LiVS(0.80)
u_4	i_3	Portr(0.80)	Exp(1.00)	Good(0.95)	Pr OK(0.90)	SV(1.00)
u_5	i_3	Portr(0.80)	Am(0.80),Exp(0.80)	Good(1.00)	Exp(0.85)	SV(1.00)
u_2	i_4	Tele(0.85)	AB(0.70),Am(0.90)	Good(0.90)	Pr OK(0.85)	LiVS(0.80)
u_4	i_4	Lands(0.90)	AB(0.80),Am(0.90)	Mod(0.90)	GD(0.90)	LiVS(0.85)
u_2	i_5	Sport(1.00)	Exp(0.95)	Good(1.00)	Exp(0.70)	SV(1.00)
u_3	i_5	Portr(0.95)	Exp(1.00)	Good(1.00)	Exp(0.80)	SV(0.70)
u_4	i_5	Sport(1.00)	Exp(0.95)	Good(1.00)	Exp(1.00)	SV(0.80)
u_1	i_6	Portr(0.95)	Exp(1.00)	Good(1.00)	Exp(1.00)	SV(0.95)
u_1	i_7	Lands(0.80)	AB(0.90),Am(0.70)	Mod(0.90)	GD(0.75)	LiVF(0.80)
u_2	i_7	Macro(0.90)	AB(1.00)	Good(0.70)	GD(0.80)	LiVF(0.90)
u_3	i_7	Sport(0.50)	AB(1.00),Am(0.75)	Good(0.75)	GD(0.90)	LiVS(0.70)
u_4	i_7	Tele(0.95)	Am(0.80)	Mod(0.80)	GD(0.60)	LiVF(0.60)
u_5	i_7	Portr(0.70)	AB(0.95)	Mod(0.75)	GD(1.00)	LiVF(0.95)
u_3	i_8	Tele(0.90)	AB(1.00)	Mod(0.90)	GD(1.00)	LiVF(0.60)
u_5	i_8	Tele(0.80)	AB(0.90)	Bad(0.80)	GD(0.95)	LiVF(0.80)
u_1	i_9	Sport(0.90)	Am(0.70),Exp(0.70)	Good(1.00)	Exp(1.00)	LiVS(0.60)
u_2	i_9	Sport(0.95)	Exp(0.80)	Good(0.90)	Pr OK(1.00)	LiVS(0.90)
u_4	i_9	Sport(1.00)	Exp(0.90)	Good(0.95)	Pr OK(0.90)	LiVS(0.80)
u_5	i_9	Sport(0.90)	Am(0.50),Exp(0.90)	Good(0.60)	Exp(0.60)	LiVS(0.70)
u_1	i_{10}	Tele(0.70)	Exp(0.75)	Good(0.75)	Pr OK(0.80)	LiVS(0.45)
u_2	i_{10}	Tele(0.85)	Exp(0.95)	Mod(0.75)	GD(1.00)	LiVS(0.90)
u_3	i_{10}	Macro(0.90)	Am(0.60),Exp(0.80)	Good(0.95)	Pr OK(0.90)	LiVS(0.70)
u_4	i_{10}	Tele(0.95)	Exp(1.00)	Mod(0.55)	Pr OK(0.95)	LiVS(1.00)
u_5	i_{10}	Macro(0.55)	Exp(0.80)	Good(0.80)	Pr OK(0.95)	SV(0.55)

Table 3.6: Items with their assigned user attribute values and corresponding supports. The column I denotes the respective items, while the column U shows the users who specified the supports. Thus, each row corresponds to one user evaluating an item.

Recommendation Requirements REQ . Users looking for recommendations need to specify their requirements regarding the items they are looking for. We denominate those requirements as *recommendation requirements* in this work. The requirements specified by users are matched against the information describing the item. Thus, recommendation requirements comprise requirements regarding *item attributes* (REQ_{ia}) and requirements regarding *user attribute values* (REQ_{uav}). For *item attributes*, users need to specify a certain value which is then used in combination with the defined *similarity measure* to match items. For *user attribute values*, users only need to specify which properties the recommended items must have. For instance, User u_1 in Table 3.7 specifies that all recommended items must be suitable for *Sport* and *Tele* photography as well as being usable by *Amateurs* with respect to photographing skills. The exact recommendation approach based on the specified requirements $REQ = REQ_{ia} \cup REQ_{uav}$ is discussed in Section 3.3. Examples of recommendation requirements specified by users are given in Table 3.7.

	REQ_{uav}	REQ_{ia}
u_1	uav_{13} (Sport), uav_{14} (Tele), uav_{22} (Amateur)	-
u_2	uav_{13} (Sport)	ia_4 (APS-C)
u_3	-	ia_1 (15), ia_4 (full)

Table 3.7: Table showing recommendation requirement examples from different users.

The relation between all previously discussed entities can be seen in the very basic UML diagram shown in Figure 3.1. The diagram is not a complete diagram of the whole system; its purpose is to demonstrate the relations (with cardinalities) between the previously mentioned entities.

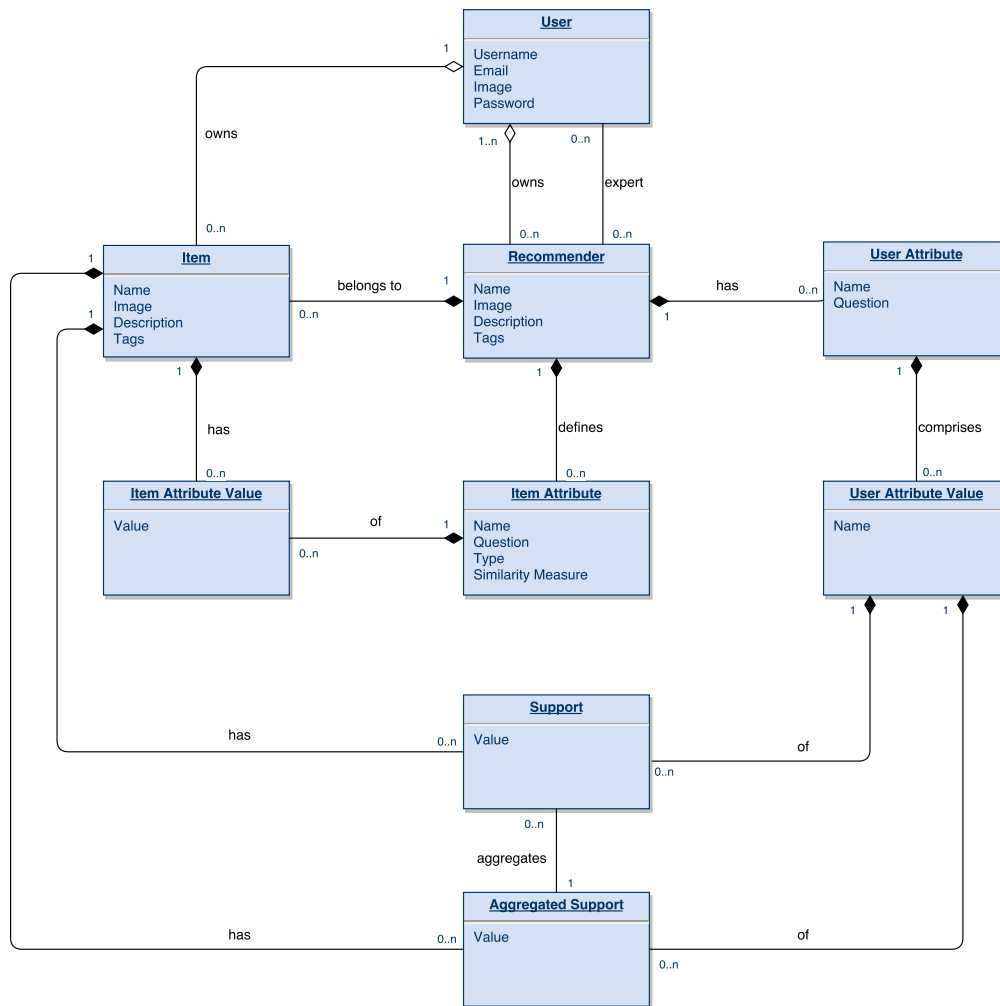


Figure 3.1: UML diagram showing the relations between the entities defined for a constraint-based recommender.

3.2 Averaging Support Values

As mentioned in the previous section, we have developed two methods to aggregate the support values specified by users. In this section, the basic approach is discussed. The more sophisticated approach using Beta Distributions is presented in Section 5.1.

The very basic idea behind this method is to average the support values for any item - user attribute value pair over all users who evaluated that specific pair. When looking at Equation (3.1), the numerator of the equation shows the sum over all support values entered for the selected item - user attribute value pair. Different to a normal average, where this sum would be divided by the number of supports entered for that selected pair, we count all users who specified a support for *any* user attribute value of the chosen user attribute ($uav \in dom(ua)$), the reason for this will be explained by a simple example

following the equations). The counting is depicted in Equation (3.2), where the brackets [...] denote a condition in *Iverson brackets*, meaning 1 is added to the sum if the condition inside the brackets is fulfilled, otherwise 0 is added (see [Knu92] for further information).

$$\text{support}_{\Sigma}(i, ua, uav) = \frac{\sum_{u \in U} s(i, u, uav)}{N_U(i, ua)} \quad (3.1)$$

$$N_U(i, ua) = \sum_{u \in U} [\exists uav \in \text{dom}(ua) \wedge s(u, i, uav) \neq \text{NULL}] \quad (3.2)$$

The reason for not using simple averaging can be shown best with an example. Using a single choice user attribute, for instance ua_1 (Field of Application, see Table 3.4) and the supports depicted in Table 3.6, only user u_1 might select Macro as the best fitting user attribute value for item i_3 , assigning a support of 100%. In contrast to that, users u_2 , u_3 , u_4 and u_5 all selected Portrait as the best fitting Field of Application for i_3 , all users assigning a support of 80%. If we simply would average the entered support values, item i_3 would have supports of 100% for Macro and 80% for Portrait although 4 out of 5 users deemed Portrait to be the better use case for camera i_3 . By using the formalism stated in Equations (3.1) and (3.2), we obtain the results shown in Equation (3.3).

$$\begin{aligned} \text{support}_{\Sigma}(i_3, ua_1, \text{Macro}) &= \frac{1.00}{5} = 0.20 \\ \text{support}_{\Sigma}(i_3, ua_1, \text{Portrait}) &= \frac{0.80 + 0.80 + 0.80 + 0.80}{5} = 0.64 \end{aligned} \quad (3.3)$$

By considering all users who assigned any user attribute value for the user attribute ua_1 , the more popular user attribute value (Portrait) now has a higher support value, although the entered supports were smaller than the one entered for Macro. All aggregated supports for the supports listed in Table 3.6 are shown in Table 3.8 using the calculations shown in this section.

The aggregated support values can then be used to select and rank items based on requirements specified by users who are looking for recommendations. The approach is discussed in Section 3.3.

	Field of Application	Experience Level	Durability	Value for Money	Loss of Value
i_1	Sport(0.925)	Am(0.213), Exp(0.913)	Mod(0.25), Good(0.713)	Pr OK(0.475), Exp(0.463)	SV(0.488), LiVS(0.45)
i_2	Sport(0.95)	AB(0.95)	Mod(0.90)	GD(1.00)	LiVF(0.90)
i_3	Macro(0.20), Portrait(0.64)	Am(0.28), Exp(0.93)	Good(0.94)	Pr OK(0.56), Exp(0.31)	SV(0.71), LiVS(0.16)
i_4	Tele(0.425), Landscape(0.45)	AB(0.75), Am(0.90)	Mod(0.45), Good(0.45)	GD(0.45), Pr OK(0.425)	LiVS(0.825)
i_5	Sport(0.667), Portrait(0.316)	Exp(0.967)	Good(1.00)	Exp(0.834)	SV(0.834)
i_6	Portrait(0.95)	Exp(1.00)	Good(1.00)	Exp(1.00)	SV(0.95)
i_7	Landscape(0.16), Macro(0.18), Sport(0.1), Tele(0.19), Portrait(0.14)	AB(0.77), Am(0.45)	Moderate(0.49), Good(0.44)	GD(0.81)	LiVS(0.14), LiVF(0.65)
i_8	Tele(0.85)	AB(0.95)	Bad(0.40), Mod(0.45)	GD(0.975)	LiVF(0.70)
i_9	Sport(0.938)	Am(0.30), Exp(0.825)	Good(0.863)	Pr OK(0.475), Exp(0.40)	LiVS(0.75)
i_{10}	Tele(0.50), Macro(0.29)	Am(0.12), Exp(0.86)	Mod(0.26), Good(0.50)	Pr OK(0.72), GD(0.25)	SV(0.11), LiVS(0.61)

Table 3.8: Aggregated supports calculated based on the values in Table 3.6

3.3 Recommendation Approach

The recommendation approach discussed in this section is used to filter and rank items according to the recommendation requirements specified by a user looking for recommendations. As mentioned in Section 3.1, we introduce two different approaches to aggregate support values. The equations shown in this section are valid using both methods for averaging support values. The aggregated support here is simply denoted as $support_{\Sigma}(i, ua, uav)$ and can be calculated with the simple approach discussed in Section 3.2 as well as with the method using Beta Distributions, which is proposed in Section 5.1. Thus, the utility function and item attribute support discussed in this section can be applied to both recommendation methods (basic and beta distribution based) implemented for this master's thesis.

The recommendation process can be divided into two phases. First, all considered items are selected into a recommended set (RS). Second, the items contained in RS are

ranked based on a utility function which we will define in this section.

To select all items RS that are relevant concerning the requirements specified by the user ($REQ = REQ_{ia} \cup REQ_{uav}$) all items that have aggregated supports larger than 0 for all user attribute values defined by the user ($uav \in REQ_{uav}$) have to be determined. Also, for all item attributes with their corresponding value specified ($ia \in REQ_{ia}$), the support needs to be larger than 0. If the user only specifies one type of attributes (either $REQ_{uav} = \emptyset$ or $REQ_{ia} = \emptyset$), items are selected based on that attribute type alone. A formal definition of the item selection is given in Equation 3.4.

$$RS = \{i \mid \forall uav \in REQ_{uav} : \text{support}_{\Sigma}(i, uav) > 0 \vee REQ_{uav} = \emptyset\} \cap \{i \mid \forall ia \in REQ_{ia} : \text{support}_{\Sigma}(i, ia) > 0 \vee REQ_{ia} = \emptyset\} \quad (3.4)$$

The necessary definition for supports regarding item attributes can be seen in Equation (3.5) where the similarity measures *Nearer is Better (NIB)*, *More is Better (MIB)* and *Less is Better (LIB)* as defined by McSherry [McS03] are used. The similarity measure *Equal is Better (EIB)* is a special case of *NIB*. Here, iav denotes the reference value specified by the user looking for recommendations, $val(i, ia)$ is the value defined for item i and item attribute ia . The functions $min(I, ia)$ and $max(I, ia)$ denote the minimum and maximum respectively for a given item attribute considering all items I .

$$\text{support}_{\Sigma}(i, ia, iav) = \begin{cases} [iav = val(i, ia)] & \text{EIB} \\ 1 - \frac{|iav - val(i, ia)|}{max(I, ia) - min(I, ia)} & \text{NIB} \\ \frac{val(i, ia) - min(I, ia)}{max(I, ia) - min(I, ia)} & \text{MIB} \\ \frac{max(I, ia) - val(i, ia)}{max(I, ia) - min(I, ia)} & \text{LIB} \end{cases} \quad (3.5)$$

As an example, the supports for the recommendation requirement of user u_3 from Table 3.7 are calculated for item i_3 . The requirements (REQ_{ia}) specified by the user state that she is looking for cameras having the sensor format *full* and more than 15 megapixels. Applying the *MIB* formula from Equation (3.5), an example can be calculated as follows for item i_3 :

$$\begin{aligned} \text{support}_{\Sigma}(i_3, ia_1, 15) &= \frac{22.3 - 18.0}{53.0 - 18.0} = 0.123 \\ \text{support}_{\Sigma}(i_3, ia_4, full) &= [ia_4 = full] = 1 \end{aligned} \quad (3.6)$$

An example list of selected items when using the recommendation requirements from user u_2 (see Table 3.7), who specified requirements with respect to user attribute values (REQ_{uav}) and item attributes (REQ_{ia}) can be seen in Table 3.9.

After all relevant items are selected, the items contained in RS need to be ranked before being presented as recommendation result to the user. Thus, we define a utility function for items with respect to the recommendation requirements REQ specified by the user. The utility function depicted in Equation (3.7) sums the aggregated supports of all item attributes and user attribute values in REQ ($ia \in REQ_{ia}, uav \in REQ_{uav}$). Additionally, Equation (3.7) contains weights which are multiplied with the summed supports. The weights $w(uav)$ and $w(ia)$ refer to item attributes and user attributes respectively. In the basic approach, no weighting is used, thus, $w(uav) = 1, \forall uav \in UAV$ and $w(ia) = 1, \forall ia \in IA$. The enhanced approach discussed in Section 5.2 optimizes this weights to enable better recommendation results.

$$\begin{aligned} \text{utility}(i, REQ) = & \sum_{uav \in REQ} \text{support}_{\Sigma}(i, uav) \times w(uav) \\ & + \sum_{ia \in REQ} \text{support}_{\Sigma}(i, ia) \times w(ia) \end{aligned} \quad (3.7)$$

By applying Equation (3.7) to the recommendation requirements specified by user u_2 in Table 3.7, we calculate the utility values for all items listed in Table 3.9. Not listed items where not included in the set based on Equation (3.4) because some support was not larger than 0. As an example, the utility for item i_1 is calculated:

$$\begin{aligned} \text{utility}(i_1, REQ) &= \text{support}_{\Sigma}(i_1, uav_{13}) \times 1 + \text{support}_{\Sigma}(i_1, ia_4) \times 1 \\ &= 0.925 \times 1 + 1 \times 1 = 1.925 \end{aligned} \quad (3.8)$$

Rank	Item	Utility
1	i_2	1.950
2	i_9	1.938
3	i_1	1.925
4	i_7	1.100

Table 3.9: Recommended items, sorted based on their respective utility. Selection of items was done using Equation (3.4). Utility was calculated using Equation (3.7)

4

PEOPLEVIEWS

The recommendation approach discussed in Chapter 3 was implemented as part of this master's thesis in an application called PEOPLEVIEWS. In this chapter, the requirements regarding the application and the chosen basic architecture that was implemented are discussed. Also, an overview of used frameworks and libraries is given. The chapter is concluded with a presentation of the web application's user interface, where several use cases of a recommender system are demonstrated, and the corresponding user interface is explained.

4.1 Architecture

When designing PEOPLEVIEWS, a list of requirements was specified.

- **Scalability:** To be able to serve a large number of users as well as offer a large number of recommenders and respective items, the application should be highly scalable.
- **Responsiveness:** The application should be very responsive to ensure a good user experience.
- **Clients:** There should at least be a web version as well as a version for mobile devices. Both versions should contribute to the same knowledge base.
- **State of the art:** The application should be developed using a state of the art architecture as well as components such that it can be supported in the future.

- **Openness:** No proprietary software or library should be used to develop PEOPLEVIEWS. The application should be deployable on any given hardware that meets its resource demand.

Because of the requirements listed, the basic architecture of PEOPLEVIEWS as depicted in Figure 4.1 was used. The system can be divided into a client and server part, which will be discussed separately.

4.1.1 PEOPLEVIEWS Server

To fulfill the requirements listed above, the PEOPLEVIEWS server application was realized as a server comprising RESTful webservice that provide methods for all operations possible in the system.

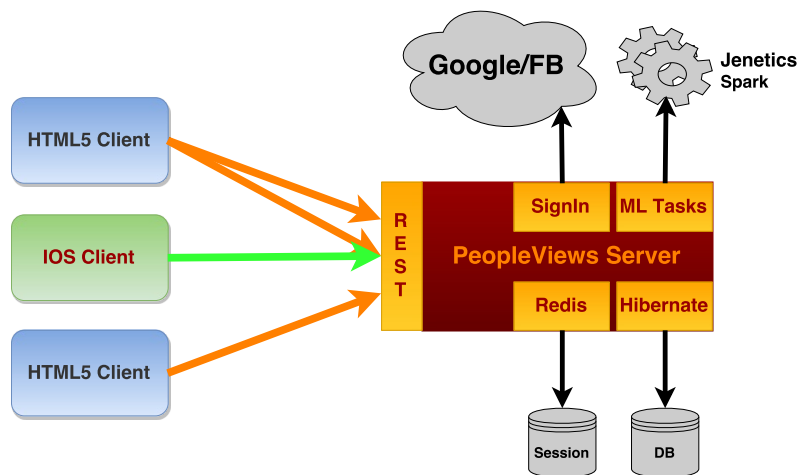


Figure 4.1: Basic architecture of PEOPLEVIEWS including all major components.

RESTful web service: A RESTful web service is based on the REST (Representational State Transfer) architecture. Rather than putting the focus on implementation details, the focus of a REST architecture is on component's roles and their interactions with each other. Moreover, in a REST architecture, everything is a resource.

A characteristic of RESTful web services is the fact, that no state is held on the server. Each web service call must identify its purpose and contain the necessary data to fulfill the request. Thus, making RESTful web services very light weight and highly scalable because load balancing can be done relatively easy.

All services are identified by their URI and an HTTP method. For instance, the URI `http://peopleviews.com/recommender/list` called with a GET request would return a list of all recommenders.

By defining the complete functionality as RESTful web services, the application can run on any arbitrary server or even cluster. Thus, fulfilling the requirements *scalability* and *responsiveness*. Also, any client that is capable of sending HTTP requests can communicate with the application server, therefore fulfilling the requirement of *multiple clients*. Also, the application was developed using a *state of the art* and *open* web development framework, which, among others, is discussed in Section 4.2.

A typical bottleneck of web applications - accessing the database - is prevented by abstracting the database access, using database connection pools and enabling an additional database cache. Also, to ensure *responsiveness*, heavy calculations (such as calculating aggregated supports) are done as background jobs on a dedicated cluster. Thus, those tasks can be moved from the main webserver if necessary. Because of the implemented web services stateless nature, sessions are kept in an in-memory storage which is used to handle authentication and session requests. The libraries used for this purpose are listed in Section 4.2.

4.1.2 Recommender as a Service

In addition to the functionality provided by RESTful web services, PEOPLEVIEWS also offers an SQL-like query language for getting recommendations. Also implemented as a RESTful web service, the syntax to get recommendations is defined as follows:

```
SELECT * FROM <RecommenderName>
WHERE <ATTRIBUTE>=<VALUE> [AND <ATTRIBUTE2>=<VALUE2> ...]
[LIMIT [<OFFSET>, ]<ITEMS>]
```

Where `<RecommenderName>` defines the desired recommender. Recommendation requirements can be specified as a list of `<ATTRIBUTE>` and `<VALUE>` pairs. Optional parameters used for pagination can be specified with `LIMIT`. The resulting ranked list of recommended items will be returned as a JSON object.

By using this query language, recommenders can be embedded into any software or service. When implementing the interface to the recommender, no knowledge of the structure and relationships of entities used in PEOPLEVIEWS needs to be known. In combination with the possibility to define any recommender using the web interface and to collect knowledge about the contained items using human computation, a recommender can be provided for any type of recommender domain. Thus, we denote this feature as providing a *Recommender as a Service (RaaS)*.

4.1.3 PEOPLEVIEWS Client

The client implemented for this master's thesis was written using HTML5 and JavaScript. All requests to the PEOPLEVIEWS server are sent via JavaScript as asynchronous HTTP requests using the previously discussed RESTful web service interface. The implemented user interface is shown in Section 4.3. The HTML5 and JavaScript libraries used to develop the web client are listed and discussed in Section 4.2.

There is also a mobile client developed for iOS devices which is being implemented as part of a bachelor thesis. This client also uses the same RESTful web service interface as the web interface. However, as this client was not part of this master's thesis, no further details about it are given in this work.

4.2 Used Software Components

To implement PEOPLEVIEWS in the previously described architecture, a couple of freely available frameworks and libraries were used. The following list provides an overview of the libraries which are essential for the developed functionality in PEOPLEVIEWS. The list includes libraries and frameworks used to develop the server side components as well as libraries and templates used in the user interface creation process.

Spring Framework⁴ is a framework for building web applications using Java EE features. The additional plugin *spring boot* allows for rapid web development. Spring supports a wide range of features which are needed for web development such as a Model-View-Controller (MVC) framework for building RESTful web services, transaction management, easy configuration management and so on. Arthur and Azadegan [AA05] discuss Springs features for rapid web development.

Hibernate⁵ is used as object-relational mapping (ORM) tool. Thus, Java objects are automatically mapped into database classes by Hibernate. Also, by using the spring data JPA plugin in combination with Hibernate, database queries can be constructed very easily in Java. Elizabeth O'Neil [O'N08] discusses the advantages of Hibernate as an ORM tool.

Redis⁶ in principle is a data structure server, basically like a database. However, in contrast to a database, Redis holds its whole dataset in memory. Thus, querying Redis is very fast, with the drawback of no provided data durability. Because of the mentioned characteristics, Redis is used to store sessions of logged in users where for a RESTful service fast response times are of utmost importance. Azat Mardan

⁴ <http://projects.spring.io/spring-framework/>

⁵ <http://hibernate.org/>

⁶ <http://redis.io/>

[Mar14] gives an overview of possible use cases of Redis for authentication and session management.

Jenetics⁷ is a Java implementation of the genetic algorithm which is discussed in Section 5.2.1 of this work. The library is written in a very modular way such that key concepts of the genetic algorithm can be configured and even exchanged with own implementations very easily. As Jenetics uses the Java Stream API, calculations can be executed in parallel quite easily.

Apache Spark⁸ is a cluster computing framework which is used in PEOPLEVIEWS for various computational expensive machine learning tasks like finding outliers (see Section 5.3). To execute machine learning algorithms with spark, the extension Apache Spark MLlib exists which provides almost all state of the art machine learning algorithms.

Google/Facebook APIs⁹ are used to login into PEOPLEVIEWS without creating a user account first. When logging in using an existing Google account, users are authenticated using their Google login data. Additionally, the user's email address is passed to PEOPLEVIEWS. When logging in using an existing Facebook account, the restfb library is used which authenticates the user using her Facebook credentials. After login, the email and profile picture stored at Facebook are provided via restfb.

Java Mail API¹⁰ is used for sending account activation emails as well as invitation emails for not yet registered users of PEOPLEVIEWS.

jQuery¹¹ is used for client-side scripting in the HTML5 client. All dynamic components of PEOPLEVIEWS web client rely on this library. JQuery is claimed to be the most used JavaScript library in use to date and provides a lot of additional plugins which were utilized in the development of PEOPLEVIEWS (most of them provide sophisticated UI components).

Bootstrap¹² is the front-end framework used to develop PEOPLEVIEWS' user interface. The framework contains HTML, CSS as well as JavaScript templates to develop dynamic web pages. The developers (initially developed by Twitter) claim that bootstrap is the most starred project on GitHub.

⁷ <http://jenetics.io/>

⁸ <http://spark.apache.org/>

⁹ <https://developers.google.com/api-client-library/java/>, <http://restfb.com/>

¹⁰ <http://www.oracle.com/technetwork/java/javamail/index.html>

¹¹ <https://jquery.com/>

¹² <http://getbootstrap.com/>

4.3 User Interface

The user interface developed for the PEOPLEVIEWS web interface is going to be shown in this section. Different scenarios, including the creation of the example recommender used throughout this work, will be described and accompanied by screenshots of the corresponding user interface. The landing page (main screen) for users not logged in is shown in Figure 4.2.

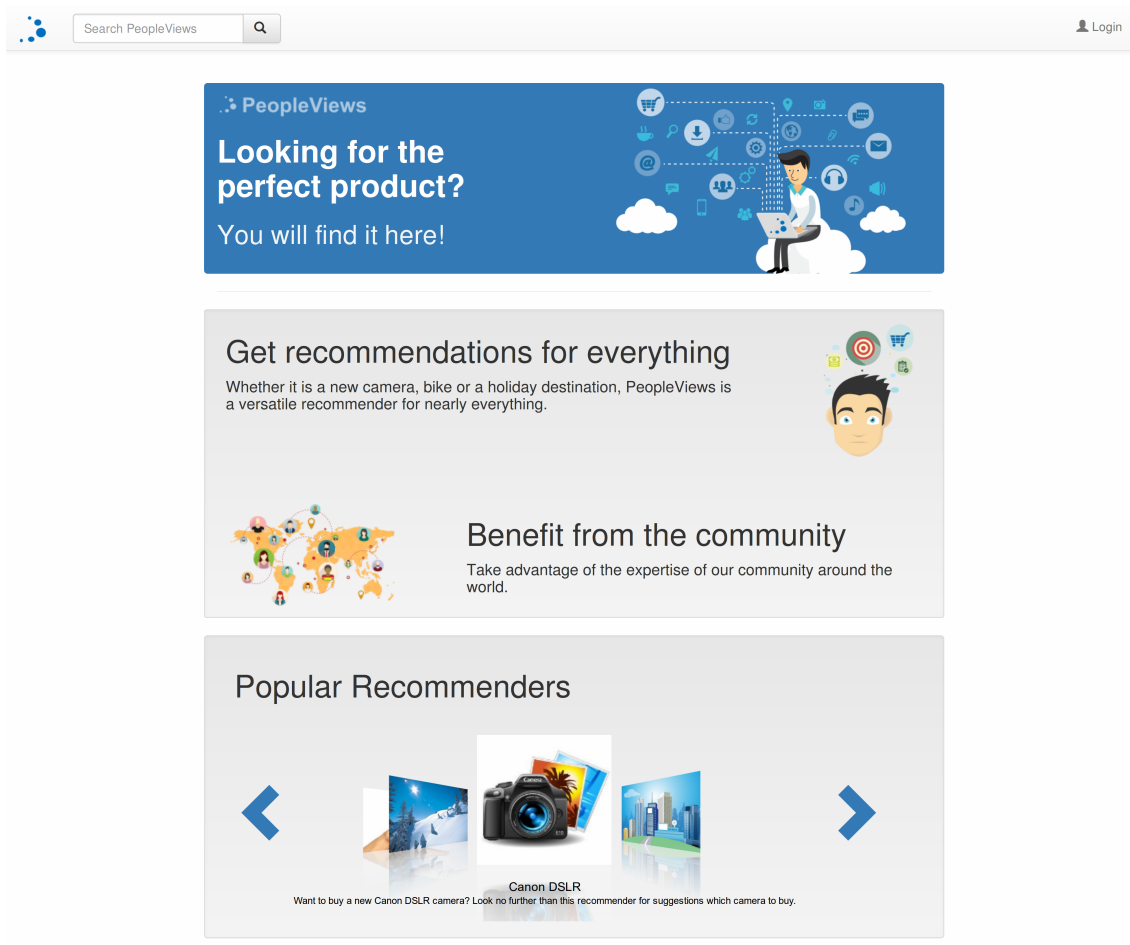


Figure 4.2: Landing page when requesting the PEOPLEVIEWS web interface. This site is also the home screen for all not logged in users.

Register and Login

To contribute to the knowledge base of PEOPLEVIEWS, users need to register with the system first. To register for a new account, users need to choose a username and password as well as specify their email address as can be seen in Figure 4.3. If the account is successfully created, users receive an activation mail containing a link to activate the

newly created account. When logging in, users also have the option to use either their Google or Facebook account, which automatically creates a new account if users log into PEOPLEVIEWS for the first time. By doing so, the email and profile picture used at the mentioned platforms is transferred to PEOPLEVIEWS. The login screen is depicted in Figure 4.4.

Create account

Username

E-mail

Password

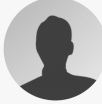
Password (Confirm)

After successfully registering for a new account, you will receive an email containing an activation link for your newly created account (Note that this could, depending on your mail provider, take arbitrarily long).

[Register](#)

Figure 4.3: Inputs presented to a user who is registering for a new account.

Login



Username

Password

[Login](#)

OR

[Sign in with Facebook](#)

[Sign in with Google](#)

[Create account](#) [Forgot password](#)

Figure 4.4: Login screen as shown to users.

Home Screen and User Profile

After successfully logging in, users are presented with the home screen illustrated in Figure 4.5. There, users are presented with a list of popular recommenders (globally most used recommenders), the users top recommenders (most used by the user), PEOPLEVIEWS points (obtained for contributing to the knowledge base) and saved recommendation requirements, denoted as saved filters. All further actions can be started using the menu on the left.

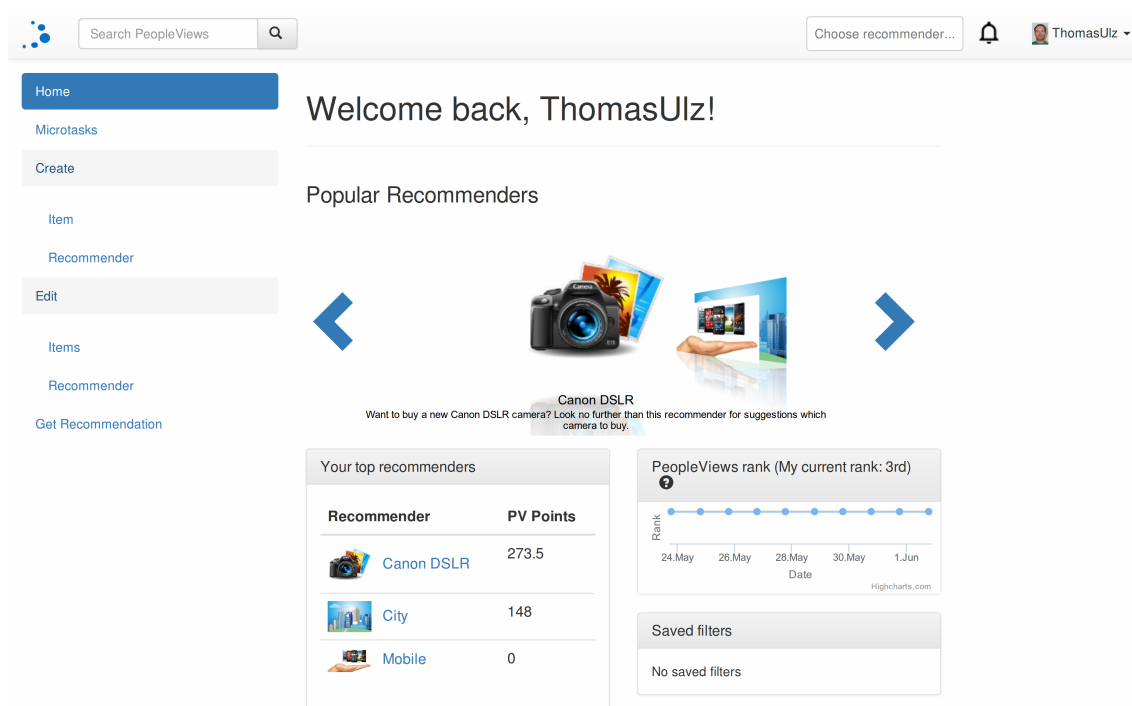


Figure 4.5: Home screen for logged in users. Information such as popular and top recommenders as well as PEOPLEVIEWS points and resulting rank are shown.


By clicking on the username shown in the top right corner and choosing *My profile* users get to their user profile where they can change their username and see a detailed list of PEOPLEVIEWS points they obtained. The user profile is shown in Figure 4.6.

Creating a New Recommender

After choosing *Create - Recommender* in the menu shown in Figure 4.5, a new recommender can be created. To do so, information such as a recommender image, name, description and tags need to be specified, as shown in Figure 4.7.

User Profile

Profile picture



Change Remove

Username

E-Mail

You've earned

423

PeopleViews points

Recommender	PeopleViews Points	Game score
City	149.5	34
Canon DSLR	273.5	24
Mobile	0	3

[Save](#)

Figure 4.6: User profile with options to change profile picture and username. Also a detailed list of PEOPLEVIEWS points if given.

Also, item attributes are defined when creating a new recommender. Figure 4.8 depicts the item attribute creation section of the add recommender view.

To conclude the creation of a new recommender, also user attributes and their corresponding user attribute values need to be specified. The associated view is shown in Figure 4.9.

Adding Items to a Recommender

To add a new item to an existing recommender, first, the desired recommender needs to be selected. If no recommender was chosen in the current session, users are asked to specify a recommender when clicking *Create - Item* as can be seen in Figure 4.10. Alternatively, users always can select their preferred recommender by specifying it in the box positioned at the top right corner of every screen, as shown in Figure 4.11.

The screenshot shows the 'Add Recommender' form in the PeopleViews application. The form is titled 'Add Recommender' and is located in the 'Recommender' section of the application. The form includes the following fields:

- Recommender name:** A text input field containing 'Canon DSLR'.
- Image:** A placeholder image of a camera with a 'Select image' button below it.
- Description:** A text area containing the text 'Want to buy a new Canon DSLR camera? Look no further than this recommender for suggestions which camera to buy.'
- Tags (comma separated):** A text input field containing 'Canon', 'DSLR', and 'Camera' as tags.

Figure 4.7: Basic information such as recommender name, image, description and tags entered when creating a new recommender.

The screenshot shows the 'Item attributes' form for the Canon DSLR recommender. The form defines three attributes:

- Megapixel:** Attribute name: Megapixel; Type: Number; Similarity measure: More is better; Question: What is the maximum Resolution of this camera in megapixels?; Usable as recommendation filter:
- Maximum ISO:** Attribute name: Maximum ISO; Type: Number; Similarity measure: More is better; Question: What is the maximum ISO value this camera can support?; Usable as recommendation filter:
- Price:** Attribute name: Price; Type: Number; Similarity measure: Less is better; Question: What is the suggested retail price of this camera?; Usable as recommendation filter:

At the bottom of the form, there is a '+ Add attribute' button.

Figure 4.8: Definition of item attributes for the Canon DSLR recommender. Attribute name, type, similarity measure and a question which is asked to users who add a new item are defined here.

After a recommender is selected, items can be added to this recommender. When creating a new item, basic data such as item name, picture, description, tags, and a link need to be specified. Also, all item attributes specified in the corresponding recommender are mandatory input fields when adding a new item. The screen illustrating the creation of a new item for the *Canon DSLR* recommender can be seen in Figure 4.12.

Figure 4.9: User attribute definitions. Users need to specify an attribute name and a question which is asked to users who evaluate the item. Also, the corresponding user attribute values are specified here.

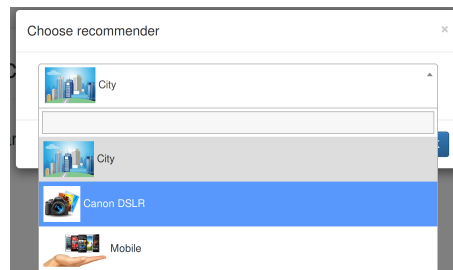


Figure 4.10: If no recommender was specified before clicking Create - Item, a pop up asks the user to specify to which recommender the new item should be added to.

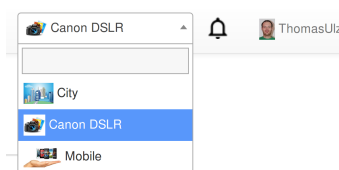


Figure 4.11: At the top right corner of every screen, a preferred recommender can be specified by the user.

Editing existing Recommenders or Items

Existing recommenders and items also can be edited after they were created. The interface to edit an existing item does not differ from the one shown in Figure 4.12 where a new item is added. The only difference is that existing values are displayed to the user to be probably changed.

Add Item

Item name

Image



Select image

Description

The Canon EOS 7D Mark II is a professional[1] digital single-lens reflex camera made by Canon.[2] It was announced on 15 September 2014 with a suggested retail price of US\$1,799.[3] Among its features are a 20.2 effective megapixel APS-C CMOS sensor, HD video recording, its 10.0 frames per second continuous shooting, viewfinder which offers 1.0× magnification with a 50 mm lens and 100% coverage, 65-point auto-focus system, a built-in Speedlite transmitter, and a new metering sensor.[2] It was preceded by the

Tags (comma separated)

sport x canon x dslr x

Link

https://en.wikipedia.org/wiki/Canon_EOS_7D_Mark_II

Item attributes

What is the maximum Resolution of this camera in megapixels?

20.2

What is the maximum ISO value this camera can support?

16000

What is the suggested retail price of this camera?

1799

Published (This item should be visible for all other users. Only published items are shown in recommendations.)

Cancel

Save

Figure 4.12: When a new item is added, basic information such as item name, image, description, tags and a link need to be specified. Also all item attributes become mandatory input fields.

When editing an existing recommender, the interface for entering the recommender information including item attribute and user attribute definitions does not differ from the interfaces shown in Figures 4.7, 4.8, and 4.9. However, the creator of a recommender also has the possibility to add additional *owners* and *experts* to the recommender. *Owners* do have the same privileges as the creator. Thus, they can edit the recommender and also add owners and experts to it. *Experts* also are allowed to edit the recommender, but are not allowed to add or remove other owners or experts. The interface for managing owners and experts is shown in Figure 4.13.

The screenshot shows the 'My Recommenders' page. At the top, there is a search bar and a user profile for 'ThomasUlz'. The left sidebar contains navigation options: Home, Microtasks (59), Create, Item, Recommender, Edit, Items, and Get Recommendation. The main content area is titled 'My Recommenders' and features a card for a 'Canon DSLR' recommender. Below the card, there are two tables. The 'Owner' table has columns for 'User', 'PeopleViews Points', and 'Action', with one entry for 'ThomasUlz' (273.5 points) and a 'Downgrade to expert' button. The 'Experts' table has columns for 'User', 'PeopleViews Points', and 'Action', with one entry for 'Michael Schwarz4' (716.5 points) and buttons for 'Remove expert' and 'Make owner'. At the bottom of the page, there are buttons for '+ Add expert' and 'Edit recommender'.



Figure 4.13: Managing owners and experts for an existing recommender. To edit the recommender data, users need to click on the Edit Recommender button at the bottom left corner of the page.

Evaluation of existing Items

To evaluate an existing item in PEOPLEVIEWS, users need to view the item's details as shown in Figure 4.15 and click the *evaluate* button. A list of items can be obtained by either searching for an item using the search box in the upper left corner of every screen as shown in Figure 4.14 or by getting a list of recommended items, which will be discussed later.

The interface used to evaluate items depicted in Figure 4.16 displays the item's information and all user attributes specified in the corresponding recommender. Here, users

The screenshot shows a web application interface for searching items. At the top, there is a search bar containing the text '7d' and a magnifying glass icon. To the right of the search bar are options for 'Choose recommender...', a notification bell, and a user profile for 'ThomasUlz'. Below the search bar is a navigation menu with 'Home' selected, and other options like 'Microtasks 65', 'Create', 'Item', 'Recommender', 'Edit', 'Items', 'Recommender', and 'Get Recommendation'. The main content area is titled 'Search result for "7d"'. It features two filters: 'Recommenders' and 'Items', both set to 'Included'. Below the filters is a table of search results:

Result	Type
 Canon EOS 7D The Canon EOS 7D is a semi-professional cropped sensor digital single-lens reflex camera made by Canon. It was announced...	Item
 EOS 7D Mark II The Canon EOS 7D Mark II is a professional[1] digital single-lens reflex camera made by Canon.[2] It was announced on 15...	Item

At the bottom of the results, there is a pagination control showing '1 - 2' with navigation arrows.

Figure 4.14: Full-text search for items. The search string will be looked for in the item's name, description and tags.

can assign supports to one or more user attribute values. If a user wishes to edit a previously done evaluation, the same steps as for evaluating the item for the first time need to be taken. All existing supports previously assigned by the user are then displayed for editing.

Microtasks

Microtasks are also used to collect evaluations from users. As mentioned in Section 3.1, microtasks are automatically assigned to users. The menu visible to users displays the number of microtasks that are assigned but not yet solved, as can be seen for instance in Figure 4.5.

A microtask regarding the user attribute *Field of Application* is shown in Figure 4.17. There, the best matching user attribute value needs to be chosen, and a support is assigned for the selected value.

Figure 4.18 depicts a CAPTCHA microtask, which should be easily solvable for all users. The reason for such microtasks is discussed in Section 5.3.

Q

Choose recommender...
🔔
👤 ThomasUlz

Home

Microtasks 65

Create

Item

Recommender

Edit

Items

Recommender

Get Recommendation

EOS 7D Mark II

← Back

Tags

sport
canon
dslr

Description

The Canon EOS 7D Mark II is a professional[1] digital single-lens reflex camera made by Canon.[2] It was announced on 15 September 2014 with a suggested retail price of US\$1,799.[3] Among its features are a 20.2 effective megapixel APS-C CMOS sensor, HD video recording, its 10.0 frames per second continuous shooting, viewfinder which offers 1.0x magnification with a 50 mm lens and 100% coverage, 65-point auto-focus system, a built-in Speedlite transmitter, and a new metering sensor.[2] It was preceded by the Canon EOS 7D.

Link

https://en.wikipedia.org/wiki/Canon_EOS_7D_Mark_II

Item attributes

Attribute	Value
Megapixel	20.2
Maximum ISO	16000
Price	1799

Evaluate
Edit

Similar items

Canon EOS 550D

Canon EOS 70D

EOS 5DSR

Figure 4.15: Viewing item details. Here, a user can choose to evaluate the item by clicking the evaluate button. At the bottom of this interface, items similar to the currently displayed item are shown.

Game

It is also possible for users to play a game where the current aggregated support of items concerning certain user attribute values needs to be guessed. The game can be invoked on the recommendation screen by clicking the *Play against the community* button, as shown in Figure 4.20. After starting the game, users are presented with an interface like the one

Evaluate »EOS 7D Mark II«

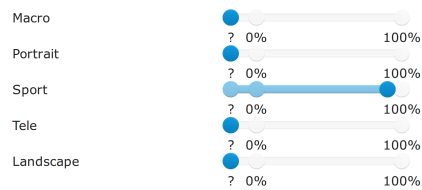
Recommender "Canon DSLR"



Description

The Canon EOS 7D Mark II is a professional[1] digital single-lens reflex camera made by Canon.[2] It was announced on 15 September 2014 with a suggested retail price of US\$1,799.[3] Among its features are a 20.2 effective megapixel APS-C CMOS sensor, HD video recording, its 10.0 frames per second continuous shooting, viewfinder which offers 1.0x magnification with a 50 mm lens and 100% coverage, 65-point auto-focus system, a built-in Speedlite transmitter, and a new metering sensor.[2] It was preceded by the Canon EOS 7D.

▾ For which field of application is this camera suited for?



▸ What is the suggested level of experience a user should have using this camera?

▸ What level of durability would you expect from this camera?

▸ What do you think is the value you get for your money when buying this camera?

▸ How fast do you think will this camera loose its value if bought new?

Cancel

Save

Figure 4.16: Interface for evaluating existing items. Users can assign supports for all specified user attribute values in the respective recommender.

Canon DSLR

Item »EOS 7D Mark II«: Which answer fits the attribute »Field of Application« best?

Macro
 Portrait

Sport
 Tele

Landscape

How well?

? 0% 100%

Don't show questions for this recommender

Skip

Next

Figure 4.17: Microtask regarding the user attribute Field of Application for the item EOS 7D MarkII.

depicted in Figure 4.19.

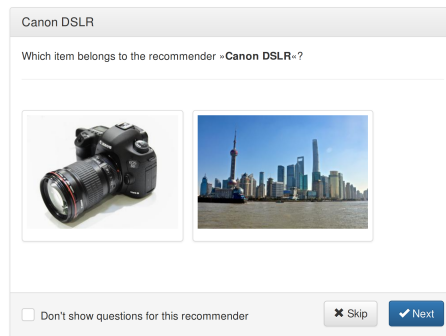


Figure 4.18: CAPTCHA microtask which should be easily solvable by all users.

Guess what the community thinks!

Recommender "Canon DSLR"

Questions answered



How well does the following question-answer combination match for »**Canon EOS 70D**«?

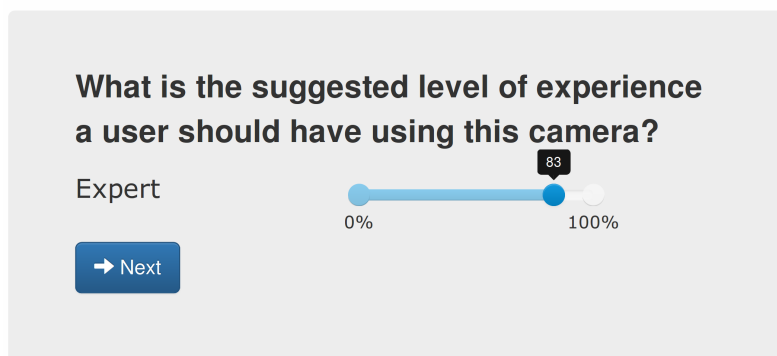


Figure 4.19: Playing a game where the current aggregated support of a given item with respect to a certain user attribute value needs to be guessed.

Getting Recommendations

Getting recommendations in PEOPLEVIEWS is possible for logged in as well as for anonymous users. The menu entry *Get Recommendations* is accessible for all users. The user can get recommendations by specifying their recommendation requirements in the list on the left-hand side of the screen. The list of recommended items is updated immediately after each change to the specified requirements. An example recommendation session can be seen in Figure 4.20. Users can save the currently specified recommendation requirements as a filter with the button in the bottom left corner of the page.

The screenshot shows the PeopleViews search results for "Canon DSLR". The interface includes a search bar at the top, a user profile for ThomasUtz, and a sidebar for refining results. The main content area displays a list of camera models with their descriptions and a "Compare" button for each item. The filter sidebar is active, showing "Sport" and "Amateur" selected under "Field of Application" and "Experience Level" respectively.

Item	Compare
Canon EOS 550D The Canon EOS 550D is an 18.0 megapixel digital single-lens reflex camera, announced by Canon on 8 February 2010.	Best in 2 attributes
Canon EOS 7D The Canon EOS 7D is a semi-professional cropped sensor digital single-lens reflex camera made by Canon. It was announced...	
Canon EOS 5D Mark III The Canon EOS 5D Mark III is a professional full-frame digital single-lens reflex (DSLR) camera made by Canon. It has a ...	Best in Sport
Canon EOS 1D X The Canon EOS-1D X is the professional flagship digital SLR camera body by Canon Inc. It succeeded the company's previous...	
EOS 1200D Canon EOS 1200D is an 18.1-megapixel digital single-lens reflex camera (DSLR) announced by Canon on 11 February 2014.	
Canon EOS 70D The Canon EOS 70D is a digital single-lens reflex camera from Canon. The EOS 70D was announced on 2 July 2013, to replac...	
EOS 80D The Canon EOS 80D is a digital single-lens reflex camera announced by Canon on February 18, 2016.[2][3] It has a body-on...	
EOS 5DSR The Canon EOS 5DS and EOS 5DS R (known as the EOS 5Ds and EOS 5Ds R in Japan) are two closely related digital SLR camera...	Worst in Amateur

Figure 4.20: List of recommended items for the requirements specified by user u_1 in Table 3.7.

If no recommendation is possible, an empty list and suggestions on which requirements to remove is shown, as is depicted in Figure 4.21.

Compare Items

From the list of recommended items, two items can be selected for comparison, as can be seen in Figure 4.20. After selecting the two items and clicking the *Compare items* button, users are presented with a comparison of items regarding the user attributes as well as the item attributes. User attributes and their user attribute values are compared using the aggregated support, which is displayed in a spider chart. The item attributes are marked based on the defined similarity measures. A comparison of two items can be seen in Figure 4.22.

The screenshot shows the PEOPLEVIEWS search interface. At the top, there is a search bar with the text "Search PeopleViews" and a magnifying glass icon. To the right of the search bar, there is a dropdown menu showing "Canon DSLR", a notification bell icon, and a user profile icon for "ThomasUlz".

Below the search bar, there is a "Refine by" section with several filter categories:

- Field of Application**
 - Macro
 - Portrait
 - Sport
 - Tele
 - Landscape
- Experience Level**
- Durability**
- Value for Money**
- Loss of Value**
 - Stable Value
 - Loses in Value Slowly
 - Loses in Value Fast

The main content area displays the search results for "Canon DSLR". At the top of this area, there is a "Back" button and two buttons: "Play against the community" and "Edit". Below this, there is a header "Item" and a "Compare" button. The main text reads "No recommendation could be found!".

A yellow callout box contains the following text: "A small suggestion for you. Remove constraint **Loses in Value Fast** to get more results".

At the bottom of the results area, there is a pagination control showing "1 - 8" and "9 - 10", and two buttons: "Compare items" and "Save filter".

Figure 4.21: If no items can be recommended, PEOPLEVIEWS suggests which constraints to remove to get items recommended again.

Comparison

[← Back](#)



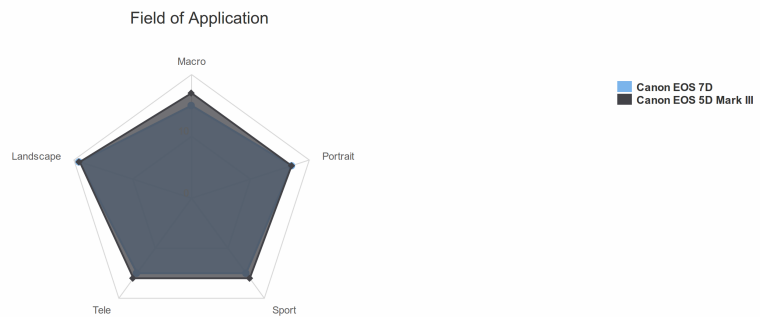
Canon EOS 7D

35



Canon EOS 5D Mark III

45



Attribute	Canon EOS 7D	Canon EOS 5D Mark III
Megapixel	18	22.3
Maximum ISO	12800	25600
Price	1699	2800

Figure 4.22: Comparing two items based on the aggregated supports for user attribute values and the values specified for the item attributes.

5

Enhanced Approaches

The method of calculating the aggregated support in the basic recommendation approach as illustrated in Chapter 3 has one major drawback. It only calculates the average *but does not weight the average in any way*. As an extreme example, a new item could only have one evaluating user who assigned a support of 90% for user attribute value *Sport*. On the other hand, item i_2 could have a vast number of evaluating users, say for instance 3,000 and an aggregated support of 85% for the same user attribute value *Sport*.

Considering this information, humans might see item i_2 as the better option concerning user attribute value *Sport*, as a large number of ratings would mean a higher confidence in the aggregated support value. However, using the approach discussed in Section 3.2, the recommendation algorithm would rank i_1 higher than i_2 based on the high rating of one user. This fact has to be seen as a drawback of the basic approach. Therefore, we introduce a more sophisticated method which utilizes *Beta Distributions*.

5.1 Beta Distribution Based Approach

Before we are going to discuss the approach itself, some basic attributes of the Beta Distribution need to be discussed. The examined characteristics will then be required to derive the Beta Distribution based approach of calculating the aggregated supports.

5.1.1 The Beta Distribution

The *Beta Distribution* is a statistical model that can be used to describe the behavior of a random variable and was used to do so in a wide variety of disciplines. As other

continuous probability distributions, the Beta Distribution is defined on the interval $[0, 1]$. It is parametrized by two (strictly positive) values, α and β which are called *shape parameters*. In this section, only the distribution's characteristics that are later used are discussed. A detailed explanation of the Beta Distribution is provided by Gupta [Gup11].

The *Probability Density Function* (PDF) of a Beta Distribution can be defined in various ways. For the calculations later shown, only the second definitions in Equation (5.1) will be needed.

$$f_X(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1} = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1} \quad (5.1)$$

For those PDFs, $0 \leq x \leq 1$ and $\alpha > 0, \beta > 0$ need to be fulfilled. In addition, the *Beta Function* $B(\alpha, \beta)$ and the *Gamma Function* $\Gamma(n)$ are defined as shown in Equation (5.2).

$$B(\alpha, \beta) = \int_0^1 u^{\alpha-1} (1-u)^{\beta-1} du \quad (5.2)$$

$$\Gamma(n) = (n-1)!$$

The shape parameters α and β are used to model the Probability Density Function of a Beta Distribution. As their name suggests, those parameters have an impact on the shape or silhouette of a Beta Distribution's PDF. One property of a probability distribution we are going to use in our approach is the so-called *mode* which corresponds to the maximum of the PDF. It, therefore, can be obtained by calculating the derivative of a given PDF and by setting the resulting derivative to zero. The maximum in a Beta Distribution, which models a random variable X gives us the most likely value for the distribution of X . The mode of a Beta Distribution can be derived as follows:

$$x^* = mode(\alpha, \beta) \Rightarrow \frac{df_X(x)}{dx} = 0 \quad (5.3)$$

$$\frac{df_X(x)}{dx} = d \left(\frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1} \right) / dx = 0 \quad (5.4)$$

By applying the chain rule for derivations, we get the derivative of $f_X(x)$.

$$\begin{aligned} \frac{df_X(x)}{dx} &= \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} (\alpha - 1) x^{\alpha-2} (1-x)^{\beta-1} \\ &+ \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} (\beta - 1) (-1) (1-x)^{\beta-2} x^{\alpha-1} = 0 \end{aligned} \quad (5.5)$$

If we then simplify the resulting equation, we obtain x^* , which equals the Beta Distribution's mode.

$$\begin{aligned}\frac{df_X(x)}{dx} &= (\alpha - 1)(1 - x) - (\beta - 1)x = 0 \\ &= x(-\alpha - \beta + 2) + \alpha - 1 = 0 \\ \Rightarrow x^* &= \frac{\alpha - 1}{\alpha + \beta - 2}\end{aligned}\tag{5.6}$$

$$\text{mode}(\alpha, \beta) = \frac{\alpha - 1}{\alpha + \beta - 2}, \quad \text{for } \alpha > 1, \beta > 1\tag{5.7}$$

As mentioned, the mode of a Beta Distribution changes depending on the values of α and β . Three different PDFs are shown in Figure 5.1, demonstrating the cases $\alpha > \beta$, $\alpha = \beta$ and $\alpha < \beta$. As can be seen there, $\alpha > \beta$ shifts the mode towards 1, meaning the most likely value described by this distribution will be closer to 1. In contrast, for $\alpha < \beta$ the mode is shifted towards 0. The third case, $\alpha = \beta$ demonstrates two facts. First, the mode will be in or near the middle of our interval $[0, 1]$ and second, the peak height at the mode will be lower than in the first two cases. Although the sum $\alpha + \beta$ is equal in all three instances, the peak will be higher while the distribution will be narrower for $\alpha \neq \beta$. *This peak height can be interpreted as confidence in the calculated most likely value.* For clear tendencies towards either 0 ($\alpha < \beta$) or towards 1 ($\alpha > \beta$) the confidence will be higher as for the case where no clear tendency can be observed ($\alpha = \beta$). The peak height can be calculated by evaluating the PDF at the calculated mode, as shown in Equation (5.8).

$$\text{peak} = f_X(\text{mode}(\alpha, \beta); \alpha, \beta)\tag{5.8}$$

5.1.2 Calculating and Scaling Aggregated Support using Beta Distribution

The two characteristics of a Beta Distribution that were discussed in Section 5.1.1 will be used to aggregate the supports in more sophisticated way than in the Basic Approach discussed in Section 3.3.

As mentioned, the mode can be seen as the most likely value of the distribution. In our case, we want the most likely value to be equal to the **average of our support values**. At first, it might seem counter-intuitive to use the average of the support values again as it was mentioned earlier, that simply averaging the supports has drawbacks. However, by

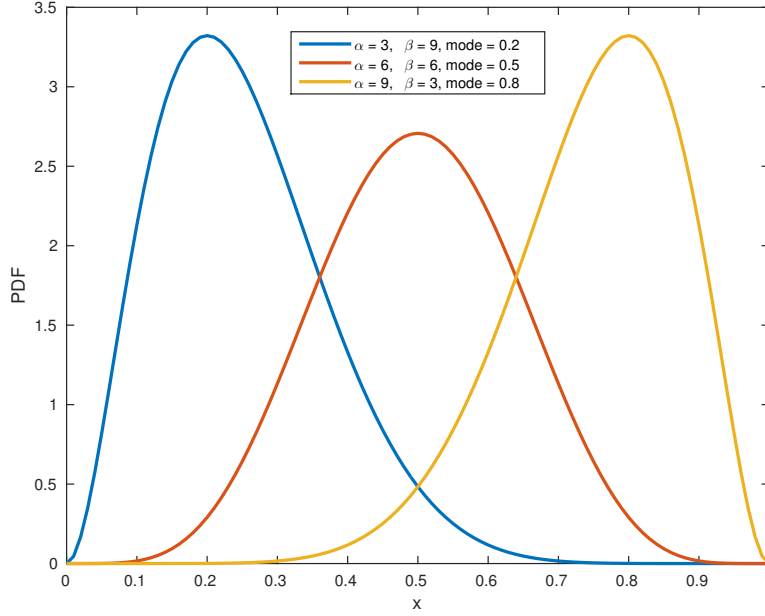


Figure 5.1: Probability Density Functions of three different Beta Distributions, demonstrating the resulting different modes and their corresponding peak heights.

using the peak height of the distribution as an additional property, those disadvantages can be overcome. By defining the equations for the shape parameters α and β as shown in Equation (5.9), we can shift the mode of the Beta Distributions as previously discussed. Additionally, by using those definitions, the mode will simplify to be equal to the average of support values.

To calculate the aggregated supports, a Beta Distribution for each item - user attribute value combination needs to be derived. Therefore, we will denote the needed parameters as $\alpha(i, uav)$ and $\beta(i, uav)$.

$$\begin{aligned}\alpha(i, uav) &:= 1 + \sum_{u \in U} s(i, u, uav) \\ \beta(i, uav) &:= 1 + N_U(i, ua) - \sum_{u \in U} s(i, u, uav)\end{aligned}\tag{5.9}$$

The total number of users evaluating a specific user attribute ($N_U(i, ua)$) needs to be calculated as in the Basic Approach.

$$N_U(i, ua) = \sum_{u \in U} [\exists uav \in \text{dom}(ua) \wedge s(u, i, uav) \neq \text{NULL}]\tag{5.10}$$

By adding one to α and β , we ensure that the prerequisites $\alpha > 1, \beta > 1$ hold, even if there is no support for a given item - user attribute value combination at all. Also, by using the definitions of the parameters α and β , the mode simplifies to be equal to the average of supports as used in our *Basic Approach*. The simplification can be seen in Equation (5.11).

$$\begin{aligned}
 \text{mode}(i, uav) &= \frac{\alpha(i, uav) - 1}{\alpha(i, uav) + \beta(i, uav) - 2} \\
 &= \frac{1 + \sum_{u \in U} s(i, u, uav) - 1}{1 + \sum_{u \in U} s(i, u, uav) + 1 + N_U(i, ua) - \sum_{u \in U} s(i, u, uav) - 2} \\
 &= \frac{\sum_{u \in U} s(i, u, uav)}{N_U(i, ua)}
 \end{aligned} \tag{5.11}$$

The peak height is used to scale the averaged support values. As discussed earlier, the peak height of a PDF is given by $peak = f_X(\text{mode}(\alpha, \beta); \alpha, \beta)$. The scaling can be seen as *confidence in the support value*. To show this by means of an example, we draw 100 and 300 samples (Φ_{100}, Φ_{300}) from a normal distribution with $\mu = 0.35$ and $\sigma = 0.1$ to simulate our support values. Both sets of samples have the same average and variance. For both sets we calculate α and β . This gives us

$$\alpha_{100} = 1 + \underbrace{\sum_{\varphi_i \in \Phi_{100}} \varphi_i}_{\approx 100 \times 0.35} \approx 36 \quad \beta_{100} = 1 + 100 - \underbrace{\sum_{\varphi_i \in \Phi_{100}} \varphi_i}_{\approx 100 \times 0.35} \approx 66 \tag{5.12}$$

for the smaller set. The larger set is described by

$$\alpha_{300} = 1 + \underbrace{\sum_{\varphi_i \in \Phi_{300}} \varphi_i}_{\approx 300 \times 0.35} \approx 106 \quad \beta_{300} = 1 + 300 - \underbrace{\sum_{\varphi_i \in \Phi_{300}} \varphi_i}_{\approx 300 \times 0.35} \approx 196 \tag{5.13}$$

If we plot the PDF of two Beta Distributions according to these values, we get the result shown in Figure 5.2 including the additional information about the peak height.

In this example, the peak height of the Beta Distribution used for the smaller set is

$$f_X(\text{mode}(\alpha_{100}, \beta_{100}); \alpha_{100}, \beta_{100}) = \frac{1}{B(36, 66)} x^{36-1} (1-x)^{66-1} = 8.3947 \tag{5.14}$$

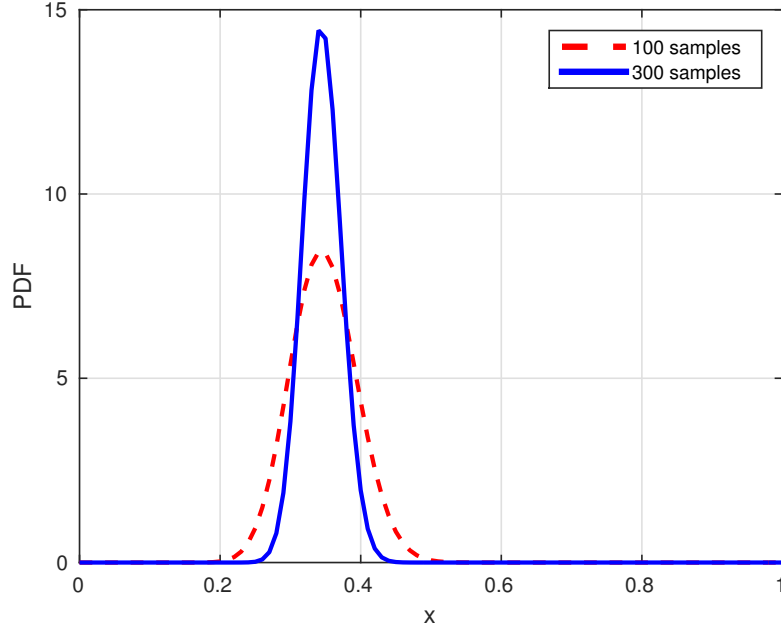


Figure 5.2: Plot showing Beta Distributions for 2 sets of samples, drawn from a normal distribution with $\mu = 0.35$ and $\sigma = 0.1$. Calculated statistical properties of both sets are $\mu_{100} = \mu_{300} = 0.344$, $\sigma_{100}^2 = \sigma_{300}^2 = 0.009$ in this plot.

and the peak height for the larger set is

$$f_X(\text{mode}(\alpha_{300}, \beta_{300}); \alpha_{300}, \beta_{300}) = \frac{1}{B(106, 196)} x^{106-1} (1-x)^{196-1} = 14.2213. \quad (5.15)$$

Hence, the larger set has a peak height of ≈ 1.75 times the peak height of the smaller set. We can then use this factor to scale the calculated average and counteract the problem described in Section 5.1. Note that by using this approach we always need to know the parameters for all items involved to be able to scale to the smallest or largest peak value. The formal description for the scaling is given by Equations (5.16) and (5.17).

$$\text{support}_{\Sigma}(i, ua, uav) = \text{mode}(\alpha(i, uav), \beta(i, uav)) \times w_m(i, uav) \quad (5.16)$$

Where $w_m(i, uav)$ is the previously mentioned scaling factor which is defined as follows.

$$w_m(i, uav) = \frac{f_X(\text{mode}(\alpha(i, uav), \beta(i, uav)); \alpha(i, uav), \beta(i, uav))}{\max_{\tilde{i} \in I} f_X(\text{mode}(\alpha(\tilde{i}, uav), \beta(\tilde{i}, uav)); \alpha(\tilde{i}, uav), \beta(\tilde{i}, uav))} \quad (5.17)$$

In this equation, \tilde{i} denotes the item with the highest peak in its PDF for the given user attribute value uav . Using these definitions and the supports specified in Table 3.6, the aggregated supports in Tables 5.1 and 5.2 are calculated. As an example, all parameters for item i_1 and user attribute value uav_{13} are calculated:

$$\begin{aligned}\alpha(i_1, Sport) &= 1 + 1.00 + 0.90 + 0.90 + 0.90 = 4.70 \\ \beta(i_1, Sport) &= 1 + 5 - 4.70 = 1.30 \\ mode(i_1, Sport) &= \frac{3.70}{4} = 0.925\end{aligned}\tag{5.18}$$

Using those values, the peak height can be calculated.

$$\begin{aligned}f_X(mode(i_1, Sport); \alpha(i_1, Sport), \beta(i_1, Sport)) \\ = f_X(0.925; 4.70, 1.30) = 2.9854\end{aligned}\tag{5.19}$$

The resulting Beta Distribution using the parameters calculated in Equation (5.18) are depicted in Figure 5.3. The mode is equal to the aggregated support calculated in Section 3.2, 0.925. The PDF at the position of the mode is 2.9854.

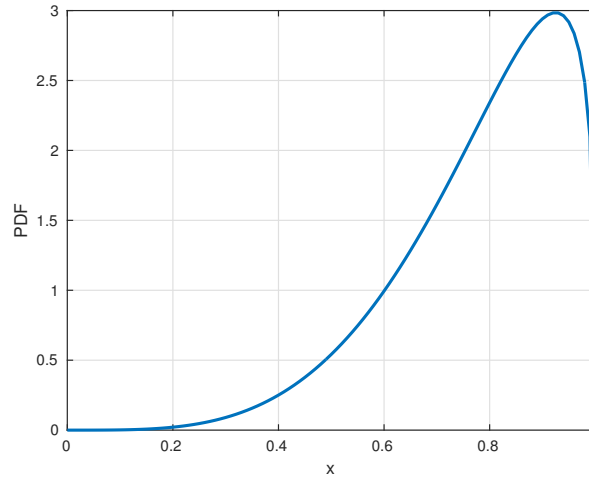


Figure 5.3: Resulting Beta Distribution for item i_1 and uav uav_{13} .

When again considering the recommendation requirements specified by user u_2 in Table 3.7, the items can be selected and ranked based on the Beta Distribution based approach. Table 5.3 displays the resulting utility values (Equation (3.7)) and the corresponding ranks, as well as the resulting ranks when using the basic approach. As an example, the utility for item i_1 is calculated.

	Field of Application	Experience Level	Durability	Value for Money	Loss of Value
i_1	Sport(4.70, 1.30, 0.925, 2.985)	Am(1.85, 4.15, 0.213, 2.206), Exp(3.75, 2.25, 0.913, 1.997)	Mod(2.00, 4.00, 0.25, 2.109), Good(3.85, 2.15, 0.713, 2.036)	Pr OK(2.90, 3.10, 0.475, 1.8770), Exp(2.85, 3.15, 0.463, 1.879)	SV(2.95, 3.05, 0.488, 1.876), LiVS(2.80, 3.20, 0.45, 1.883)
i_2	Sport(1.95, 1.05, 0.95, 1.719)	AB(1.95, 1.05, 0.95, 1.719)	Mod(1.90, 1.10, 0.90, 1.579)	GD(2.00, 1.00, 1.00, 2.000)	LiVF(1.90, 1.10, 0.90, 1.579)
i_3	Macro(2.00, 5.00, 0.20, 2.4576), Portrait(4.20, 2.80, 0.64, 2.1105)	Am(2.40, 4.60, 0.28, 2.234), Exp(5.65, 1.35, 0.93, 3.403)	Good(5.70, 1.30, 0.94, 3.556)	Pr OK(3.8, 3.20, 0.56, 2.050), Exp(2.55, 4.45, 0.31, 2.179)	SV(4.55, 2.45, 0.71, 2.214), LiVS(1.80, 5.20, 0.16, 2.634)
i_4	Tele(1.85, 2.15, 0.425, 1.512), Landscape(1.90, 2.10, 0.45, 1.505)	AB(2.50, 1.50, 0.75, 1.654), Am(2.80, 1.20, 0.90, 2.035)	Mod(1.90, 2.10, 0.45, 1.505), Good(1.90, 2.10, 0.45, 1.505)	GD(1.90, 2.10, 0.45, 1.505), Pr OK(1.85, 2.15, 0.425, 1.512)	LiVS(2.65, 1.35, 0.825, 1.793)
i_5	Sport(3.00, 2.00, 0.667, 1.778), Portrait(1.95, 3.05, 0.316, 1.796)	Exp(2.90, 1.10, 0.967, 3.071)	Good(4.00, 1.00, 1.00, 4.00)	Exp(3.50, 1.50, 0.834, 2.109)	SV(3.50, 1.50, 0.834, 2.109)

Table 5.1: Relevant parameters for Beta Distribution calculated based on supports in Table 3.6 for items i_1 to i_5 . The values listed in brackets are in correct order: α , β , mode, $f_X(\text{mode})$

First, all items need to be selected to determine the maximum peak value for scaling. The items i_1, i_2, i_7 and i_9 fullfill $\text{support}_{\Sigma}(i, uav_{13}) > 0$ and $\text{support}_{\Sigma}(i, ia_4) > 0$. The supports regarding item attributes are not scaled. The maximum peak for $\text{support}_{\Sigma}(i, uav_{13})$ is 3.133. Knowing this, the utility for i_1 can be calculated:

$$\begin{aligned}
 \text{utility}(i_1, REQ) &= \text{support}_{\Sigma}(i_1, uav_{13}) \times 1 + \text{support}_{\Sigma}(i_1, ia_4) \times 1 \\
 &= 0.925 \times \frac{2.985}{3.133} + 1 = 1.881
 \end{aligned} \tag{5.20}$$

	Field of Application	Experience Level	Durability	Value for Money	Loss of Value
i_6	Portrait(1.95, 1.05, 0.95, 1.719)	Exp(2.00, 1.00, 1.00, 2.00)	Good(2.00, 1.00, 1.00, 2.00)	Exp(2.00, 1.00, 1.00, 2.00)	SV(1.95, 1.05, 0.95, 1.719)
i_7	Landscape(1.80, 5.20, 0.16, 2.634), Macro(1.90, 5.10, 0.18, 2.538), Sport(1.50, 5.50, 0.1, 3.055), Tele(1.95, 5.05, 0.19, 2.496), Portrait(1.70, 5.30, 0.14, 2.747)	AB(4.85, 2.15, 0.77, 2.358), Am(3.25, 3.75, 0.45, 2.046)	Moderate(3.45, 3.55, 0.49, 2.038), Good(3.20, 3.80, 0.44, 2.050)	GD(5.05, 1.95, 0.81, 2.496)	LiVS(1.70, 5.30, 0.14, 2.747), LiVF(4.25, 2.75, 0.65, 2.122)
i_8	Tele(2.70, 1.30, 0.85, 1.858)	AB(1.90, 1.10, 0.95, 2.320)	Bad(1.80, 2.20, 0.40, 1.522), Mod(1.90, 2.10, 0.45, 1.505)	GD(2.95, 1.05, 0.975, 2.553)	LiVF(2.40, 1.60, 0.70, 1.593)
i_9	Sport(4.75, 1.25, 0.938, 3.133)	Am(2.20, 3.80, 0.30, 2.015), Exp(4.30, 1.70, 0.825, 2.334)	Good(4.45, 1.55, 0.863, 2.507)	Pr OK(2.90, 3.10, 0.475, 1.877), Exp(2.60, 3.40, 0.40, 1.907)	LiVS(4.00, 2.00, 0.75, 2.109)
i_{10}	Tele(3.50, 3.50, 0.50, 2.037), Macro(2.45, 4.55, 0.29, 2.214)	Am(1.60, 5.40, 0.12, 2.885), Exp(5.30, 1.70, 0.86, 2.747)	Mod(2.30, 4.70, 0.26, 2.278), Good(3.50, 3.50, 0.50, 2.037)	GD(2.00, 5.00, 0.25, 2.458), Pr OK(4.60, 2.40, 0.72, 2.234)	SV(1.55, 5.45, 0.11, 2.965), LiVS(4.05, 2.95, 0.61, 2.082)

Table 5.2: Relevant parameters for Beta Distribution calculated based on supports in Table 3.6 for items i_6 to i_{10} . The values listed in brackets are in correct order: α , β , mode, $f_X(\text{mode})$

Rank	Item	Utility	Rank Basic	Utility Basic
1	i_9	1.938	2	1.938
2	i_1	1.881	3	1.925
3	i_2	1.521	1	1.95
4	i_7	1.097	4	1.10

Table 5.3: Items ranked using aggregated supports calculated using the Beta Distribution based approach as well as the utility obtained using the basic approach.

As can be seen in Table 5.3, the selected items are now ranked differently than when using the basic approach. Because the confidence in supports for items with fewer evaluations is lower, the utility for these items will be scaled down, and thus, the items are not ranked as high as without using this scaling.

The advantage of using this method to scale the supports is the non-linear nature of the probability density function's peak height. If, for instance, the ratio of the number of ratings would be used, the scaling would be too extreme. An item having three times as many ratings as another item, already would be scaled by a factor of three. In the example shown in Figure 5.2, the factor between 100 and 300 ratings however is only 1.75. Having 10 times as many ratings as another item would then result in a scaling factor of roughly three, instead of 10 when using the ratio of number of ratings.

5.2 User Preferences

When defining the utility function used in both proposed recommendation approaches (Equation (3.7)), a weight regarding user attribute values ($w(uav)$) and a weight regarding item attributes ($w(ia)$) was introduced but set to 1 for both approaches. However, by changing these weights according to user preferences, an improvement in recommendation accuracy can be achieved. To learn the weights, the genetic algorithm is used.

5.2.1 Genetic Algorithm

Genetic algorithm is a method used in machine learning or artificial intelligence to solve optimization or search problems. The basic idea of the algorithm is to use the process of natural selection. The main parts of the algorithm can be categorized into the following parts:

1. **Initialization:** An initial population size is specified which depends on the nature of the problem. Then, the initial population is created randomly in most cases.

2. **Evaluation and Selection:** Based on a fitness or utility function which measures the quality of the current generation, parts of the existing population are selected for *breeding*.
3. **Crossover and Mutation:** Using the previously selected sub-population, a new generation is generated using genetic crossover and mutation operations. A simple example is depicted in Figure 5.4.
4. **Termination:** Steps 2 and 3 are repeated until a specified stopping criterion is met or the defined maximum number of generations is reached, the algorithm is stopped, and the fittest population is given as result.

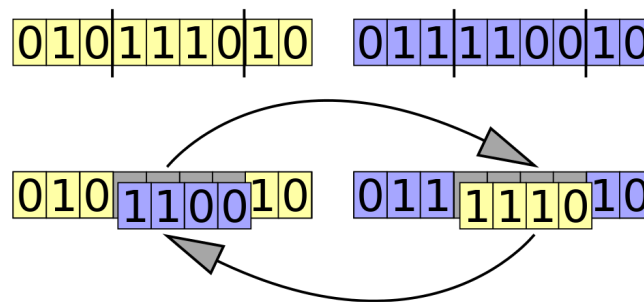


Figure 5.4: Basic principle of genetic algorithm's crossover and mutation step. Figure from wikimedia.org, released under CC license.

After initialization, steps 2 and 3 are repeated until the termination criterion specified is met. A detailed explanation of the genetic algorithm is given for instance by Mitchell in [Mit98].

5.2.2 Learning of User Preferences

To learn the weights associated with user attribute values and item attributes a *fitness function* needs to be specified. Because the goal of this optimization task is to increase the recommendation quality as perceived by the users, the same metric as used for evaluating the algorithms (Chapter 6) is used.

By using this approach, two different types of user preferences can be learned.

1. **Global User Preferences** are calculated considering all users who already assigned a user attribute value to any item. Using this information it is possible to learn for example that users value the attribute *Sport* of user attribute *Field of Application* higher than *Macro*, although both user attribute values were selected. Thus, when calculating the utility of items for a query containing both user attribute values,

$w(Sport) > w(Macro)$ will ensure that the global user preferences are honored.

Also, because global user preferences are not bound to a certain user, the learned preferences can also be used for newly registered users and users who only use the system to get recommendations without ever evaluating any item.

2. **Personalized User Preferences** can be calculated using the same approach as was used for the global user preferences. However, instead of considering all users only the respective user's contributions to the system are taken into account when calculating the personal user preferences.

To calculate personal user preferences, a user needs to have a large number of interactions with the system before something meaningful can be learned about the user's preferences. Therefore, it is reasonable to specify a threshold of evaluations a user needs to reach for a given recommender before calculating his personal user preferences. However, because personal user preferences are considered to be more accurate for a given user, they overwrite the global user preferences for a user if personal user preferences exist. Clearly by definition, personal user preferences can not be used for newly registered users or users who only use the system to get recommendations without ever evaluating any item.

As the evaluation in Section 6.3.2 is showing, a major improvement in recommendation accuracy can be obtained if user preferences are calculated and used to weight aggregated supports when calculating the item rankings.

5.3 Dataset Quality Assurance

As the recommendation accuracy and therefore quality relies on the information provided by users, bad data quality is a problem. Bad data quality is mostly caused by input of users which are either uninformed about the item they are evaluating or by malicious users. Both cases are considered as being *outliers* in our knowledge base. A definition of outliers and their identification was given by Hawkins [Haw80]. In his work, Hawkins notes that to detect outliers, models of normal behavior need to be defined.

One approach for such models is to measure the time users take to evaluate items. These models are created based on already existing user interactions with the system. New interactions are then compared against the models and users are *rated* based on their behavior in the system. Also, CAPTCHA microtasks are used to find uninformed or malicious users. The so-called *human score* denotes the probability of a user being human and is calculated mostly based on the obtained models and the answers given to CAPTCHA microtasks. The lower this score gets, the more probable it is that the user

interacting with the system might just be a malicious bot. If a user's behavior deviates too much from the calculated models, she is classified as a bot (human score = 0) and every input is completely ignored. Each user's human score is then used to weight the user's input when calculating the aggregated supports. Therefore, Equation (3.1) for the basic support needs to be adapted:

$$\text{support}_{\Sigma}(i, ua, uav) = \frac{\sum_{u \in U} s(i, u, uav) \times hs(u)}{N_U(i, ua)} \quad (5.21)$$

Where $hs(u)$ is the human score of the respective user. This score considers all human score *points* a user got in the QA when using PEOPLEVIEWS. Points are awarded for correct answering CAPTCHAs and for confirming to timing and behaviour models when interacting with the system. The sum of points is weighted with the human scores of the top users, such that it will be in the range [0, 1]. The formal definition can be seen in Equation (5.22).

$$hs(u) = \min \left(1, \left(\sum_{p \in \text{points}} p \right) \cdot \frac{\sum_{u \in \text{topuser}} hs(u)}{|\text{topuser}|} \right) \quad (5.22)$$

To correctly account for the human score, also the calculation of the shape parameters for the Beta Distribution based approach from Equation 5.9 needs to be changed accordingly:

$$\begin{aligned} \alpha(i, uav) &:= 1 + \sum_{u \in U} s(i, u, uav) \times hs(u) \\ \beta(i, uav) &:= 1 + N_U(i, ua) - \sum_{u \in U} s(i, u, uav) \times hs(u) \end{aligned} \quad (5.23)$$

The total number of users evaluating which is used in both approaches is also scaled down using the human score:

$$N_U(i, ua) = \sum_{u \in U} [\exists uav \in ua \wedge s(u, i, uav) \neq NULL] \times hs(u) \quad (5.24)$$

For an evaluation of the impact of quality assurance on the recommendation quality, the reader of this work is referred to the master's thesis of Michael Schwarz. The focus of his thesis is on quality assurance, therefore a detailed evaluation will be done as part of his work.

5.4 Additional Features

In addition to the approaches discussed in detail in this section, there are also features which will be shortly discussed here. These features do not improve the recommendation accuracy but add additional features to the recommender system which are useful concerning security or user convenience.

5.4.1 Obfuscating

As users contribute to the knowledge base of a constraint-based recommender, they enter information which reflects their personal opinions and tastes. Thus, exposing this possibly sensitive information needs to be prevented. Also, when being able to infer internal information, users can sabotage the recommender by manipulating the supports of certain items.

Imagine the evil user u_6 knows that user u_5 surely has evaluated item i_2 . If user u_6 now evaluates the same item and knows the recommendation approach, she can - based on the change in aggregated supports - a) guess how many people already rated the item and b) in the case of item i_2 also calculate the supports user u_5 specified.

As an example, user u_6 assigns a support of 0 to all user attribute values for the user attribute *Field of Application*. By observing that the aggregated support changes from 0.95 to 0.475 she now knows that only one user has rated that item and that the specified support for *Sport* by that user was 0.95. Thus, we obfuscate the supports specified by users as suggested for example by Kandappu et al. [KFBS14] and Parameswaran et al. [PB07a]. Both papers state that obfuscating the user input in a correct way does not influence the recommendation accuracy. To not affect the accuracy of our recommender, a random value (between -5 and 5 is) added to the supports specified by users, which results in a mean of 0 when only considering the added values. Thus, the aggregated supports are not changed by obfuscating the user inputs. Of course, the resulting support value is bounded to stay in the range [0.0, 1.0].

For simplicity and understandability reasons, this enhancement is not considered in the examples computed in this work.

5.4.2 Recommendation Explanation

As mentioned in Chapter 1, one advantage of constraint-based recommenders is the possibility to *explain* the recommendations shown to a user.

When considering the recommendation requirements (*REQ*) specified by a user, items are filtered based on those requirements as discussed in Section 3.3. For all requirements

$req \in REQ$, the remaining items are marked accordingly as being the highest rated or lowest rated item for the particular requirement req . One example for the requirements specified by user u_2 in Table 3.7 can be seen in Figure 4.20.

Also, if a user specifies requirements which lead to an empty result set because no item can satisfy all specified criteria, a suggestion is given on which requirement to remove to get results again. Do generate the suggestions a diagnosis algorithm proposed by Felfernig and Schubert [FSZ12] in a modified form is used. In our implementation a set of constraints is considered as *inconsistent* if the resulting set of recommended items is empty. An example of a generated diagnosis can be seen in Figure 4.21.

5.4.3 Similar Items

When viewing the details of an item, users are presented with a list of items similar to the one they are currently viewing. The list of similar items is generated using the so-called *cosine-based similarity*, which is a concept of item-item collaborative filtering, as discussed by Sarwar et al. [SKKR01].

To calculate similarities between items, they are represented as vectors in an n -dimensional space, where n in our case is the number of user attribute values in a recommender. The similarity between two item vectors \vec{i} and \vec{j} is then calculated as shown in Equation (5.25) where \cdot denotes the dot product between the two vectors.

$$sim(\vec{i}, \vec{j}) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\|_2 \times \|\vec{j}\|_2} \quad (5.25)$$

To get a vector representation of an item, the supports for all user attribute values are stored in a vector. The ordering of supports is determined by the ID of user attribute values. Using the example defined in Table 3.4, an item vector would comprise the supports in the order as illustrated in Equation (5.26). If a certain aggregated support value does not exist because the respective user attribute value was never assigned to the item, an aggregated support of 0 is assumed in the item vector.

$$v_{i_1} = \begin{bmatrix} \text{support}_{\Sigma}(i_1, ua_1, uav_{11}) \\ \vdots \\ \text{support}_{\Sigma}(i_1, ua_1, uav_{15}) \\ \vdots \\ \text{support}_{\Sigma}(i_1, ua_5, uav_{51}) \\ \vdots \\ \text{support}_{\Sigma}(i_1, ua_5, uav_{53}) \end{bmatrix} \quad (5.26)$$

6

Evaluation

In this chapter, the implemented approaches of this thesis are evaluated and compared against other baseline methods. Since most well known recommendation approaches are not directly compatible with constraint-based recommenders, we use *random*, *most-frequent* and *case-based* as baseline methods. The reason most algorithms are not compatible with the constraint-based recommendation process is that most approaches either use a single (star) rating assigned to items (collaborative-filtering) or the items information (content-based) to calculate recommendations. In a constraint-based recommender system, however, more data is available which can be used in the recommendation process. Also, as mentioned in Section 1.1, other approaches do not include any inconsistency management. If no recommendation can be found, most other algorithms (including the baseline approaches) are not able to find and suggest a possible solution on what requirements to change to get recommendations again.

We are going to shortly introduce the three used baseline methods in this chapter. Additionally the possible improvements when learning user preferences will be evaluated in this chapter as well. Some parts of this evaluation were already published in [FUH⁺15]. Also in this chapter, the dataset used for the evaluation and its acquisition process are discussed.

6.1 Dataset

To evaluate the proposed algorithms for constraint-based recommenders, a suitable dataset is needed. As mentioned, standard recommendation approaches can not be directly applied to constraint-based recommendation settings. For this reason, no suitable

dataset exists which can be used unmodified to evaluate our algorithms. The used dataset was collected as part of the work done for this master's thesis.

6.1.1 Acquisition

The data collection to gather the dataset used to evaluate the algorithms was done using the wiki-based platform WEEVIS¹³. The wiki-based style allows for fast creation of knowledge-based recommenders. In addition to that, the defined recommenders can be used without the need for user accounts. WEEVIS was chosen to collect the dataset because the recommendation approaches were developed and evaluated in a phase when the PEOPLEVIEWS user interface was not finished. Therefore, PEOPLEVIEWS could not be used. Also, the simple user interface of WEEVIS is well known by participants because of the wiki style and no account is needed to participate in the data acquisition process.

To collect data, a Canon DSLR recommender was created in WEEVIS. Users participating in the study were asked to first specify requirements regarding cameras and after that select the item that they think matches those requirements best, irrespective of the item's position in the list of recommended items. The data sample of specified requirements plus the selected best matching item were stored as one data sample for the dataset. Figure 6.1 is showing WEEVIS as used for the data acquisition process.

The screenshot shows the WEEVIS interface for the 'PeopleViews Canon DSLR Recommender'. The page title is 'PeopleViews Canon DSLR Recommender'. The interface is divided into two main sections: 'Questions' and 'Solutions'.

Questions:

- Field of Application: Macro
- Experience Level: amateur
- Durability: no answer
- Value for Money: no answer
- Loss of Value: no answer

Solutions:

Solutions	Support
Canon EOS 50D	100%
Canon EOS 5D Mark III	100%
Canon EOS 60D	100%
Canon EOS 6D	100%
Canon EOS 70D	100%
Canon EOS 7D	100%
Canon EOS 7D Mark II	100%

Figure 6.1: Data acquisition process using WEEVIS. Users specified their requirements and selected the best matching camera.

¹³ <http://www.weevis.org/>

6.1.2 Data

The collected dataset comprises the contributions of 356 distinct user sessions. The distribution of evaluations over the existing items, as done by the users, can be seen in Figure 6.2. WEEVIS offers the possibility to export all collected user interactions, therefore the dataset could then be used to evaluate the different recommendation approaches. However, one assumption needs to be made with respect to support values. As can be seen in Figure 6.1, WEEVIS offers no possibility to specify supports regarding the selected user attribute values. Therefore, a support of 1 is assumed for every value chosen by the user (0 for not chosen values).

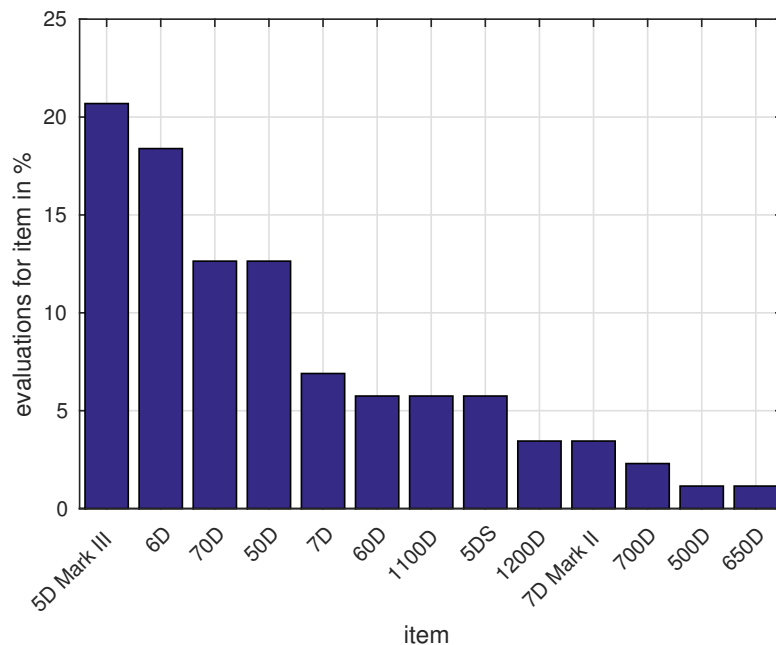


Figure 6.2: Distribution of evaluations over existing items in collected dataset.

6.2 Baseline Methods and Evaluation Approach

The three baseline methods used to compare our approaches against are *random*, *most frequent*, and *case-based (user similarity)*. These rather simple recommendation approaches are applied in this evaluation because other, more sophisticated approaches can not be used directly with the data contained in a constraint-based recommender.

6.2.1 Random

The most trivial recommender algorithm utilized in this section, the *random* recommender does not consider the recommendation requirements (*REQ*) specified by the user at all. Instead, all available items are returned in random order for each request to the recommender.

6.2.2 Most Frequent

The second baseline algorithm, *most frequent* also does not consider the recommendation requirements (*REQ*) specified by users looking for recommendations. Instead, the list of all items is sorted by the total number of distinct users who assigned any user attribute value to that item (see Equation (3.2)). For example, considering the values entered in Table 3.6, all items would be ranked as shown in Table 6.1.

Rank	Item	N_U
1	i_3	5
2	i_7	5
3	i_{10}	5
4	i_1	4
5	i_9	4
6	i_5	3
7	i_4	2
8	i_8	2
9	i_2	1
10	i_6	1

Table 6.1: Items ranked using the most frequent approach, based on data from Table 3.6

6.2.3 Case-Based

The last baseline algorithm used for comparison, *case-based* recommendation, is based on previous user interactions. The items recommended to a user are then based on the items selected by other, *similar* users. For our dataset where users specified user attribute values and the best matching item, user similarity is determined based on the specified values. Consider the subset shown in Table 6.2 and the requirements uav_{13} (Sport), uav_{23} (Expert), uav_{42} (Price is OK) for a camera specified by user u_4 . If the similarity metric *EIB* is applied for the attributes, the items would be ranked as stated in the table. Based on that metric, user u_1 specified the most similar requirements to the ones specified by user u_4 , therefore the item selected by u_1 is ranked first when calculating recommendations. Because similar user profiles are used to rank the items, this approach

will be denoted as *User Similarity* in all evaluation plots.

User	Specified User Attributes	Selected Item	Rank
u_1	uav_{13} (Sport), uav_{23} (Expert), uav_{42} (Price is OK)	i_1	1
u_2	uav_{11} (Macro), uav_{23} (Expert), uav_{42} (Price is OK)	i_3	2
u_3	uav_{13} (Sport), uav_{22} (Amateur), uav_{41} (Good Deal)	i_4	3

Table 6.2: Example data as collected for this evaluation using WEEVIS.

6.2.4 Evaluation Approach

To evaluate the implemented approaches, the recommenders need to be trained first. Therefore, the used datasets are randomly split into train and testsets. After training the recommender using the train set, the performance is evaluated using the evaluation metric illustrated in Section 6.2.5.

To test if the algorithms generalize to a dataset, cross-validation is done when evaluating the different algorithms. Because of the dataset's size and the assumption that the recommender will always be in a state where all available data is used for training, a *leave-one-out* cross-validation was performed. In that approach, in each round one of the data samples is kept out of the training set and used as single test-case. The process is repeated for all entries in the dataset, such that every data sample is used for testing once. The results are then averaged over all performed evaluation runs.

6.2.5 Evaluation Metric

To evaluate the performance of the implemented approaches, an evaluation metric needs to be defined. In [SG10] Shani et al. define *Prediction Accuracy* and state that systems that provide a higher prediction accuracy will be preferred by users. Therefore, a measure for prediction accuracy that can be applied to our recommender system needs to be found.

In our evaluation, prediction accuracy will be discussed by analysing the *recall* of all algorithms. Cremonesi et al. [CKT10] in their work describe how to evaluate *top N recommenders*. Top N in this context means that in the evaluation process it is checked if the item contained in the testset is recommended in the first N positions of all recommended items. One single run of the leave-one-out cross-validation process would therefore contain the following steps:

1. Select one data sample as testset.
2. Train the recommender algorithms with the remaining items from the dataset.

3. Use requirements specified in the testset's data sample to get recommendations.
4. Check if item of the testset's data sample is contained in the first N recommended items. If so, a *hit* is recorded, otherwise no hit will be counted.

Therefore, for one run, we can either have 0 or 1 hit for the respective testset. To calculate the overall (average) recall and precision, the formulae from Cremonesi et al. [CKT10] are used. In Equations (6.1), (6.2), and (6.3), N denotes the number of items considered when checking if the testset's data sample resulted in a hit. For instance, if the first three items are considered, N would be set to three.

$$recall(N) = \frac{\#hits(N)}{|D|} \quad (6.1)$$

$$precision(N) = \frac{recall(N)}{N} \quad (6.2)$$

$$f\text{-measure}(N) = 2 \times \frac{precision(N) \times recall(N)}{precision(N) + recall(N)} \quad (6.3)$$

Here, $\#hits(N)$ is the overall count of hits recorded when doing leave-one-out cross-validation while $|D|$ is the overall size of the dataset used for the evaluation process.

6.3 Evaluation Results

The evaluation results for the two proposed recommendation approaches will be discussed in this section. Additional means of improving the recommendation quality such as user preferences are discussed separately.

6.3.1 Recommendation Approaches

The recommendation approaches are compared against the three baseline methods random, most frequent, and case-based. The used evaluation metrics and the evaluation approach are explained in Section 6.2. In addition to comparing recall for different N as discussed in Section 6.2.4, also the relative improvement compared to a chosen baseline method is shown in these evaluations. The relative improvement with respect to recommendation accuracy is defining how much more likely it is to find the desired item in the top N recommended items, compared to the chosen baseline method.

Results

The evaluation results in terms of recall for N in the range $[1, 10]$ are shown in Figure 6.3. In this figure it can be seen that both implemented methods, the basic approach and the Beta Distribution based approach, outperform all three baseline algorithms. The Beta Distribution based approach achieves a recall of nearly 100% already when considering the first five recommended items, whereas the basic approach also has a recall of 100% when checking the first seven items.

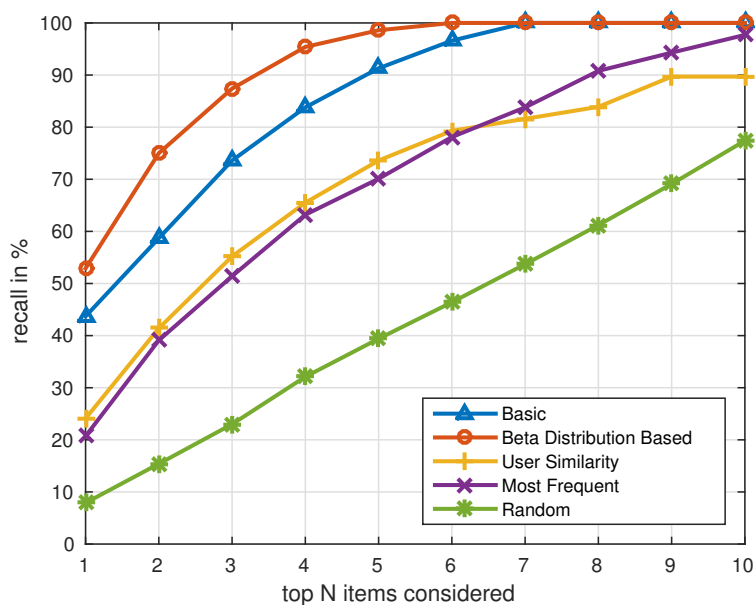


Figure 6.3: Recall for N in the range $[1, 10]$ for all four considered algorithms. The two implemented algorithms outperform all three baseline algorithms for every N .

To highlight the results achieved by the two algorithms implemented as part of this master's thesis, the relative improvements of those two algorithms over the baseline method most frequent are plotted in Figure 6.4. The basic approach achieves an improvement of at least 25% over most frequent when considering the cases *top 1* to *top 5*. The Beta Distribution based approach achieved an even better result, showing an improvement of at least 40% for the cases *top 1* to *top 5*.

When looking at the improvement for the special case *top 1*, the relative improvement of both algorithms is over 100% compared to most frequent. This means, that for both algorithms it is twice as likely that the desired item is ranked first, as for the most frequent algorithm.

In Figure 6.5, the precision according to Equation (6.2) is plotted for N in $[1, 10]$. Also for this measure, the basic and Beta Distribution based approaches outperform the

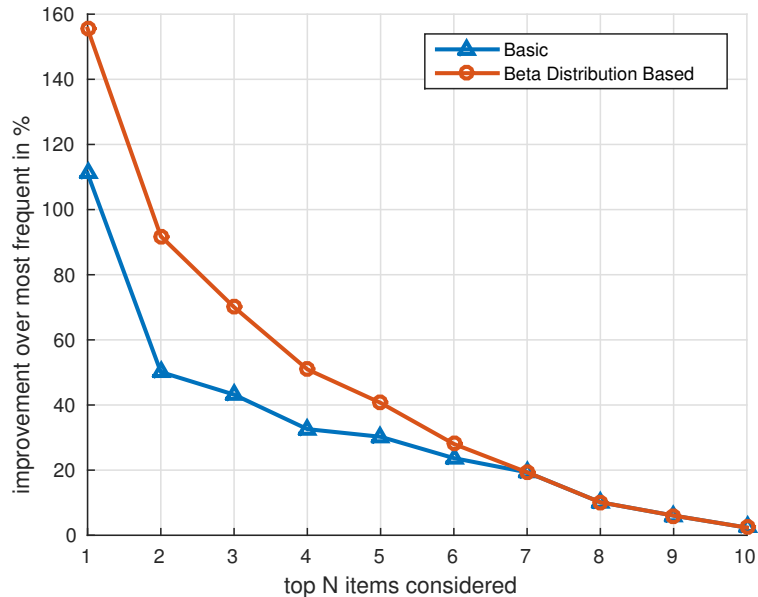


Figure 6.4: Relative improvement of both implemented approaches compared to most frequent. For the cases top 1 to top 5 very high improvements can be seen.

considered baseline methods. Because the recall shown in Figure 6.3 converged to 100% for all methods except random and user-similarity, also the precision converges to the same value for the other three algorithms. The results depicted in Figure 6.5 show a constant precision for the random approach, because only for that algorithm the increase in recall, as can be seen in Figure 6.3, is linear. The precision of the other four approaches is decreasing, because the increase in recall flattens at some point.

Concluding, the f-measure or F_1 -score for N in $[1, 10]$ is plotted for all five considered algorithms in Figure 6.6. As can be seen in Equation (6.3), the f-measure can be interpreted as being an average of recall and precision. As with the other two discussed measures, recall and precision, also when evaluating the algorithms with respect to the f-measure the two algorithms implemented in this master's thesis achieve better results than the three baseline methods.

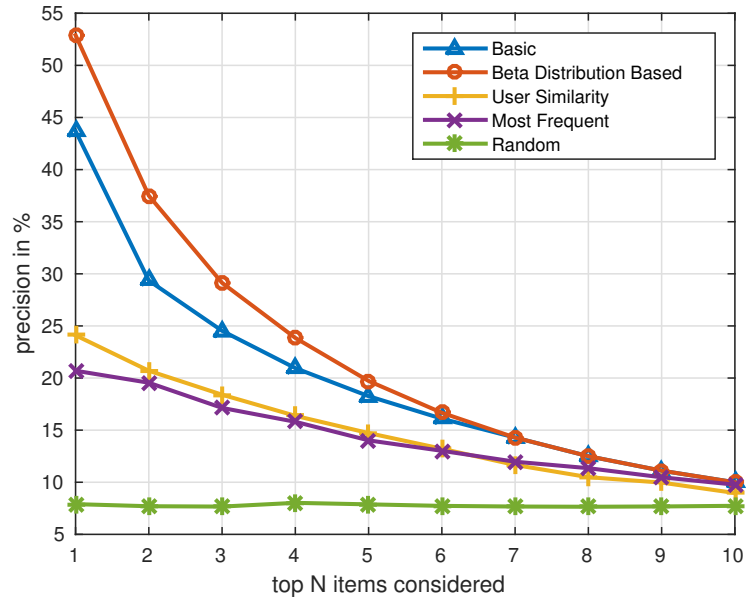


Figure 6.5: Precision for N in the range $[1, 10]$ for all five considered algorithms. The two implemented algorithms outperform all three baseline algorithms for every N .

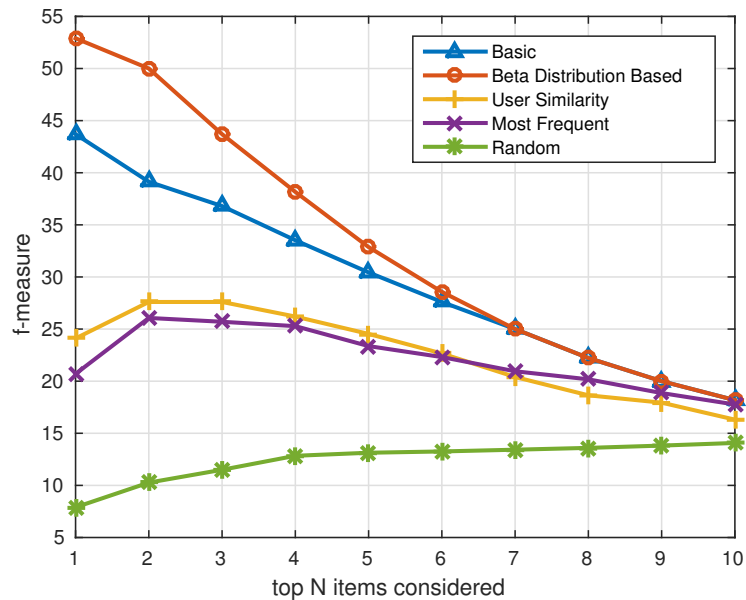


Figure 6.6: F-measure or F_1 -score for N in the range $[1, 10]$ for all five considered algorithms. The two implemented algorithms outperform all three baseline algorithms for every N .

6.3.2 User Preferences

In Section 5.2 the learning of *user preferences* was discussed. Learning those preferences has a massive impact on the recommendation quality and therefore the achieved results will be discussed in this section.

As explained in Section 5.2, user preferences can be learned on a *per-user* basis or globally for all users in the system. Although it is obvious that learning the weights for each individual user will result in a higher improvement than learning system wide weights, the second approach was chosen for this evaluation. The reason for this is that a lot of user interaction in the system is needed to learn individual preferences, however, when collecting the data most users just used the recommender for one or two recommendation sessions.

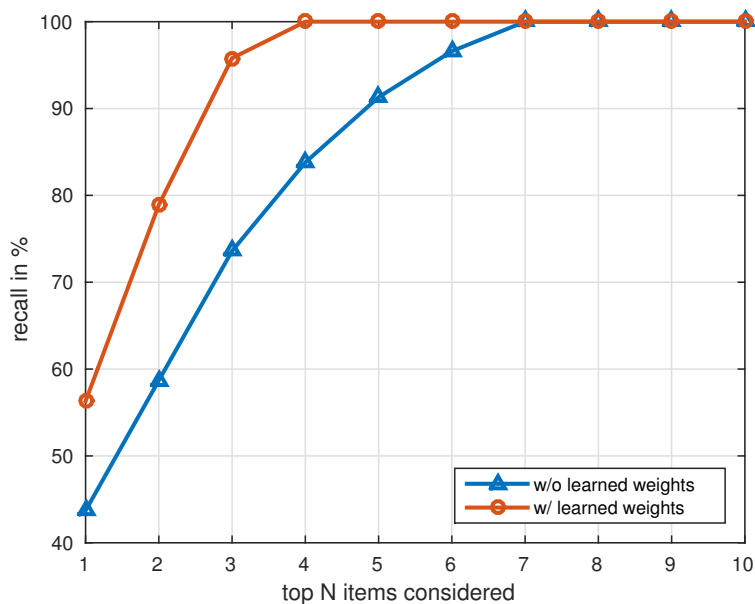


Figure 6.7: Recall for N in the range $[1, 10]$ for both considered approaches. The approach where user preferences are learned outperforms the basic approach without learning any weight for every N .

The evaluation was done using the basic recommendation approach, discussed in Section 3.2 as the adaptive weighting can be seen more easily in the formulae discussed in that section. Also the impact of learning weights can be seen more easily for that algorithm.

In the evaluation, the fitness function for the genetic algorithm was set to the respective recall function, $recall(N)$. This results in optimal weights for each evaluated case *top 1* to *top 10*. The results depicted in Figure 6.7 show the recall for the approach using

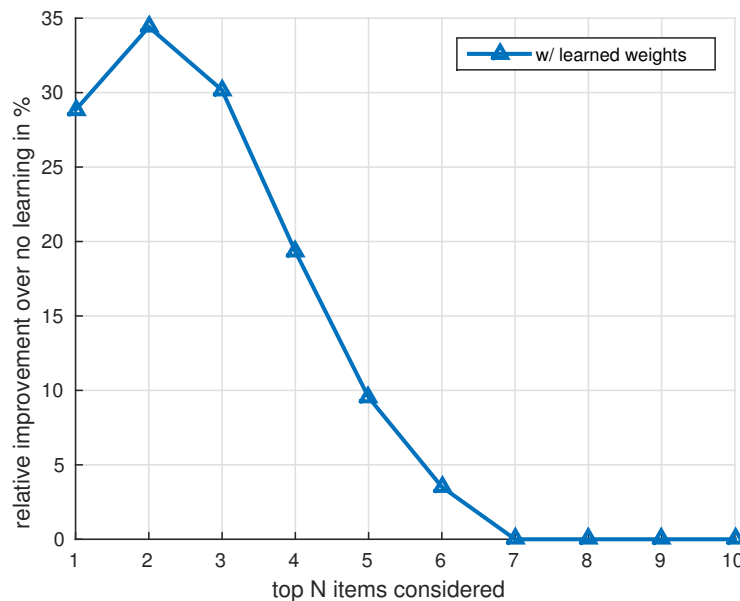


Figure 6.8: Relative improvement when learning weights compared to not learning user preferences at all. An improvement of up to 35% was achieved for this approach.

learned weights as well as for the method without learning any user preferences. It can be seen that the approach with optimal weights for each case (*w/ learned weights*) clearly outperforms the case where no learning was applied (*w/o learned weights*).

The relative improvement of the approaches where user preferences are learned compared to not learning weights at all is depicted in Figure 6.8. There, an improvement of up to 35% for learning weights can be seen.

The precision plotted for N in $[1, 10]$ for the two considered approaches is shown in Figure 6.9. Also for this metric, the approach where user preferences are learned outperforms the basic approach where no weights are learned.

Concluding, also the f-measure or F_1 -score is plotted for N in $[1, 10]$ for the two previously discussed approaches. As can be seen in Figure 6.10, the approach where weight learning is applied, also achieves a higher f-measure than the approach where no user preferences are learned.

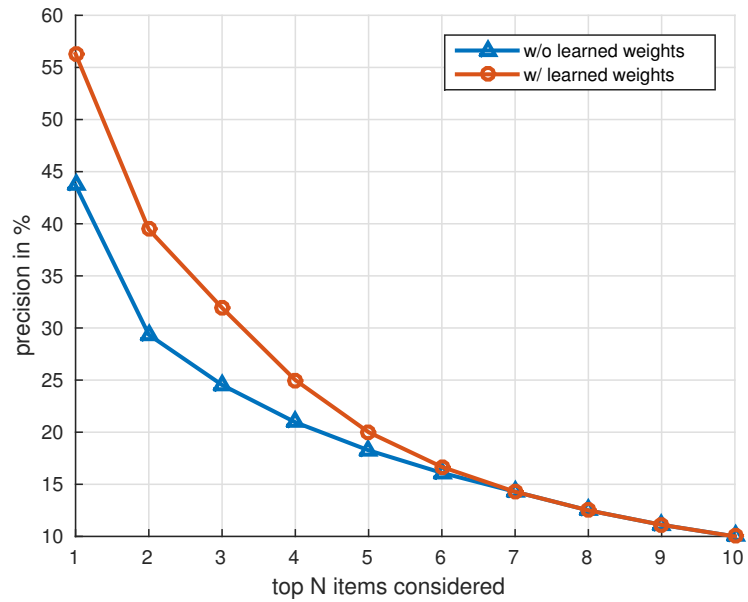


Figure 6.9: Precision for N in the range $[1, 10]$ for both considered approaches. The approach where user preferences are learned outperforms the basic approach without learning any weight for every N .

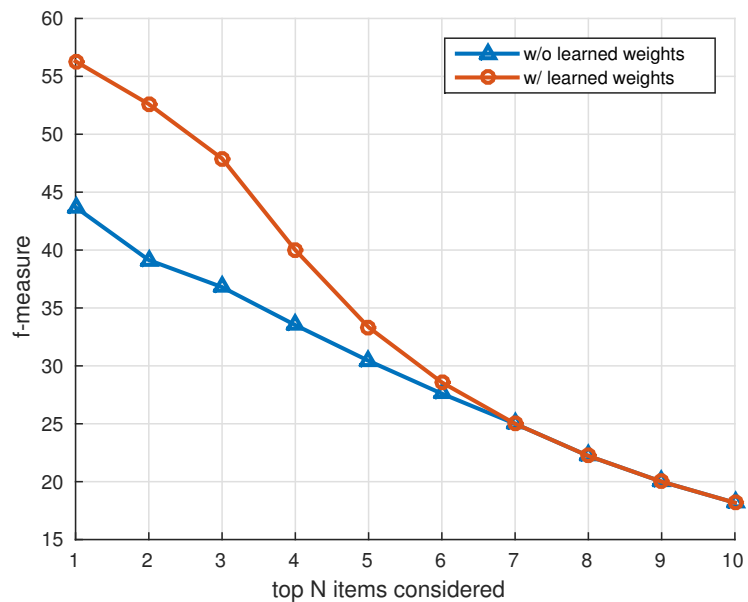


Figure 6.10: F -measure or F_1 -score for N in the range $[1, 10]$ for both considered approaches. The approach where user preferences are learned outperforms the basic approach without learning any weight for every N .

6.3.3 Other Improvements

The improvements discussed in Section 5.4 do not have any impact on the recommendation quality. Their purpose is to either complicate attacks aimed at the recommendation results or to improve user experience when using the recommender. Therefore, no evaluation regarding the recommendation accuracy will be done for those improvements.

7

Limitations and Future Work

In this chapter, the limitations of the work done within the context of this master's thesis are discussed. Also, future work, which might deal with some of the mentioned limitations, as well as future enhancements are listed.

7.1 Limitations

The limitations of this work are going to be split into different categories. For each category, the restrictions are then discussed. The mentioned limitations are of technical and algorithmic nature, as well as limitations in the evaluation of our approaches.

Technical

Technical limitations of the system that result from the chosen programming language, frameworks and technologies are listed here.

1. Java: PEOPLEVIEWS (as mentioned in Section 4.1) is implemented using Java and the Spring framework. Thus, the need to install the Java runtime before deploying PEOPLEVIEWS arises. Also, case studies by Peter Sestoft [Ses10] show that Java might be slower than other comparable languages for numeric computations although the difference is not very large.
2. HTML5: Because many of PEOPLEVIEWS' UI components use HTML5, not all browsers are capable of displaying the UI correctly. Microsoft's IE, for instance, does not support HTML5 very well. Browsers with no problems include Chrome, Firefox, Edge, and Safari.

Algorithmic

For the algorithms discussed in this master's thesis, limitations are listed in this section.

1. Recommendation speed: When calculating recommendations, aggregated supports are used to rank the items. For the recommender to be responsive, those values need to be calculated *before* a recommendation is requested. The need for background tasks which update all aggregated supports whenever a user interacts with the system arises. In a resource constraint system, this could lead to performance problems.
2. Genetic algorithm: By learning user preferences by using genetic algorithm, the recommendation approach becomes even more computationally expensive. On resource constraint hardware, this could lead to poor performance of the system.

Evaluation

There are also limitations with respect to the evaluations done in Chapter 6.

1. Baseline methods: As well known and sophisticated recommendation algorithms are not directly applicable for a constraint-based recommender, we used random, most frequent and case-based as baseline methods in our evaluations. Those algorithms, however, are not very sophisticated, and a more complex recommendation approach adapted for constraint-based recommenders might produce better results than the three methods that were used.
2. Size of dataset: The dataset used for evaluating the algorithms comprises contributions from 356 participants. The average number of evaluations per item, therefore, is around 30. For machine learning algorithms like *genetic algorithm* as used to calculate user preferences, such small data samples might lead to *overfitting*. Hawkins [Haw04] gives a good overview of the problem of overfitting.
3. Assumption of Supports: Because the platform used for the data acquisition process (WEEVIS) did not allow to input supports, a support of 1 for every chosen user attribute value and 0 for every not chosen value was assumed in the evaluation process, as already mentioned in Section 6.1.2.

7.2 Future Work

Some of the limitations listed in Section 7.1 could be solved by further improving the current state of PEOPLEVIEWS or by doing additional work in the context of this project. This section lists both, future work which aims to improve the limitations listed in Section 7.1 as well as possible enhancements.

Recommendation Algorithm

The two approaches introduced in this master's thesis basically rely on the same principle: Aggregating the specified supports of users calculating a recommendation based on those values. Future work in the context of recommendation algorithms includes the adoption of existing, more sophisticated recommendation approaches which can not be directly applied to a constraint-based recommender system.

For instance, *matrix factorization*, as discussed by Koren et al. [KBV09], is known to produce good results in *collaborative filtering* recommender systems. In its simplest form, matrix factorization is applied to a sparse, 2-dimensional *rating* matrix containing the ratings specified by users for certain items. If adopted for a constraint-based recommender system, the matrix needs to have more dimensions, as there possibly are more than one rating per item (in our case as many supports as user attributes are possible).

Machine learning approaches are used to learn user preferences and thus, improve the recommendation accuracy. Also, outliers are found using machine learning techniques. In [AT05] Adomavicius and Tuzhilin state that *artificial neural networks* or *decision trees* can be used to generate recommendations in a content-based recommender system. The point the authors make is that machine learning algorithms do not use a heuristic to calculate the recommendations. Instead, a model based on the underlying data is built. Therefore, for instance applying neural networks to a constraint-based recommender should be examined as future work.

Also, meta information such as names, descriptions, and tags that are entered for items are not used in the recommendation process. Pazzani and Billsus in their work [PB07b] discuss aspects of content-based recommendation systems, which use the items' descriptions and a user profile to generate recommendations. In PEOPLEVIEWS, a user profile is generated but only used to assign microtasks to users. The already existing user profile in combination with the mandatory item descriptions could be used to at least improve the recommendation accuracy of the approaches introduced in this work.

As was discussed in Section 2.5, hybrid approaches might mitigate drawbacks of recommendation algorithms while combining their advantages. Therefore, hybrid approaches which combine two or more recommendation algorithms should be evaluated.

Dataset

For a dataset to be usable with more sophisticated machine learning approaches, large amounts of data need to be collected. Therefore, a long-term data collection process needs to be initiated, where a large dataset can be collected. It is also necessary to use PEOPLEVIEWS for that data acquisition process to also collect information such as supports.

Clients

As part of this master's thesis, an HTML5 web interface which also can be used on mobile devices was developed. Also, a native iOS client was implemented in the context of a bachelor thesis. To support more platforms, at least a native Android version needs to be developed as well.

8

Conclusion

In this work, the terminology and entities for a constraint-based recommender system which relies on human computation were defined. Using those two concepts, a web-based system capable of handling the configuration of an arbitrary number of new recommender domains was implemented. Thus, human computation can be used for the creation and maintenance of recommenders and the items included in those recommenders, as well as for the collection process of information needed to generate recommendations. By using the "wisdom of the crowd", new recommenders can be built up faster and more easily. Also, information aggregated from a large number of user inputs will likely better match new users preferences.

To calculate recommendations, two approaches are presented in this master's thesis. Both algorithms use recommendation requirements specified by users looking for recommendations to match against a knowledge base. Those requirements, seen as constraints, lead to the selection of recommended items. For ranking the items, two different approaches were proposed. The performance of those two algorithms was evaluated in comparison to three baseline methods. The evaluation results clearly show that both algorithms produce far superior results than the baseline algorithms. In addition to the two recommendation approaches further enhancements which can be applied to both methods were discussed. These additional components further improve the recommendation quality as shown in the evaluation.

The implemented user interface PEOPLEVIEWS features a client-server architecture with an interface realized as RESTful web services. Therefore, it is easy to extend for future requirements. Also, any client that can send HTTP requests is able to communicate with the recommender systems backend. Thus, the implemented system can be viewed

as a Recommender as a Service (RaaS), which is a novel concept. As a proof of concept, an iOS client was developed by a student as her bachelor thesis.

Concluding, PEOPLEVIEWS and the included algorithms is a highly configurable recommender system that is capable of providing users the ability to define their own recommenders. The proposed algorithms were shown to deliver exceptional performance improvements of up to 100% in certain cases when compared with the considered baseline methods. To improve user experience, the implemented algorithms also allow to find inconsistencies in user input and assist the users in finding a solution for which recommendations can be calculated.

Bibliography

- [AA05] John Arthur and Shiva Azadegan. Spring framework for rapid open source j2ee web application development: a case study. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2005 and First ACIS International Workshop on Self-Assembling Wireless Networks. SNPD/SAWN 2005. Sixth International Conference on*, pages 90–95. IEEE, 2005.
- [AC10] Deepak Agarwal and Bee-Chung Chen. flda: matrix factorization through latent dirichlet allocation. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 91–100. ACM, 2010.
- [AHX15] Ahmad Abdel-Hafez and Yue Xu. Exploiting the beta distribution-based reputation model in recommender system. In *AI 2015: Advances in Artificial Intelligence*, pages 1–13. Springer Science + Business Media, 2015.
- [AT05] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.
- [BFG11] Robin Burke, Alexander Felfernig, and Mehmet H Göker. Recommender systems: An overview. *AI Magazine*, 32(3):13–18, 2011.
- [BGLB15] Joeran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger. Research-paper recommender systems: a literature survey. *Int J Digit Libr*, jul 2015.
- [BHC⁺98] Chumki Basu, Haym Hirsh, William Cohen, et al. Recommendation as classification: Using social and content-based information in recommendation. In *AAAI/IAAI*, pages 714–720, 1998.
- [BK07] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75–79, 2007.
- [BL07] James Bennett and Stan Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.
- [BNJ03] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

- [BS97] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.
- [Bur99] Robin Burke. Integrating knowledge-based and collaborative-filtering recommender systems. In *Proceedings of the Workshop on AI and Electronic Commerce*, pages 69–72, 1999.
- [Bur00] Robin Burke. Knowledge-based recommender systems. *Encyclopedia of library and information science*, 69(Supplement 32):180–200, 2000.
- [Bur02] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.
- [CHH13] Chen-Yao Chung, Ping-Yu Hsu, and Shih-Hsiang Huang. β p: A novel approach to filter out malicious rating profiles from recommender systems. *Decision Support Systems*, 55(1):314–325, apr 2013.
- [CKT10] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010.
- [CLMP99] Michelle Keim Condliff, David D Lewis, David Madigan, and Christian Posse. Bayesian mixed-effects models for recommender systems. In *ACM SIGIR’99 Workshop on Recommender Systems: Algorithms and Evaluation*, volume 15. Citeseer, 1999.
- [CP12] Li Chen and Pearl Pu. Critiquing-based recommenders: survey and emerging trends. *User Modeling and User-Adapted Interaction*, 22(1-2):125–150, 2012.
- [DMM⁺07] Casey Dugan, Michael Muller, David R Millen, Werner Geyer, Beth Brownholtz, and Marty Moore. The dogear game: a social bookmark recommender system. In *Proceedings of the 2007 international ACM conference on Supporting group work*, pages 387–390. ACM, 2007.
- [ERK11] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction*, 4(2):81–173, 2011.
- [FB08] Alexander Felfernig and Robin Burke. Constraint-based recommender systems: technologies and research issues. In *Proceedings of the 10th international conference on Electronic commerce*, page 3. ACM, 2008.
- [FFJZ06] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. An integrated environment for the development of knowledge-based

- recommender applications. *International Journal of Electronic Commerce*, 11(2):11–34, 2006.
- [FFJZ11] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Developing constraint-based recommenders. *Recommender systems handbook*, 1:187, 2011.
- [FFJZ15] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Constraint-based recommender systems. In *Recommender Systems Handbook*, pages 161–190. Springer Science+Business Media, 2015.
- [FHN⁺14] Alexander Felfernig, Sarah Haas, Gerald Ninaus, Michael Schwarz, Thomas Ulz, Martin Stettinger, Klaus Isak, Michael Jeran, and Stefan Reiterer. Rec-turk: Constraint-based recommendation based on human computation. In *RecSys 2014 CrowdRec Workshop*, pages 1–6, 2014.
- [FJN⁺14] Alexander Felfernig, Michael Jeran, Gerald Ninaus, Florian Reinfrank, Stefan Reiterer, and Martin Stettinger. Basic approaches in recommendation systems. In *Recommendation Systems in Software Engineering*, pages 15–37. Springer, 2014.
- [FJS⁺15] Alexander Felfernig, Michael Jeran, Martin Stettinger, Thomas Absenger, Thomas Gruber, Sarah Haas, Emanuel Kirchengast, Michael Schwarz, Lukas Skofitsch, and Thomas Ulz. Human computation based acquisition of financial service advisory practices. *Organizational Support*, page 27, 2015.
- [FSZ12] Alexander Felfernig, Monika Schubert, and Christoph Zehentner. An efficient diagnosis algorithm for inconsistent constraint sets. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 26(01):53–62, 2012.
- [FUH⁺15] Alexander Felfernig, Thomas Ulz, Sarah Haas, Michael Schwarz, Stefan Reiterer, and Martin Stettinger. Peopleviews: Human computation for constraint-based recommendation. In *ACM RecSys 2015 CrowdRec Workshop*, 2015.
- [GN04] Arjun K Gupta and Saralees Nadarajah. *Handbook of beta distribution and its applications*. CRC Press, 2004.
- [GNOT92] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.
- [Gup11] Arjun K. Gupta. Beta distribution. In *International Encyclopedia of Statistical Science*, pages 144–145. Springer Science + Business Media, 2011.

-
- [Haw80] D. M. Hawkins. *Identification of Outliers*. Springer Science + Business Media, 1980.
- [Haw04] Douglas M. Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, Jan 2004.
- [HKTR04] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.
- [JI02] Audun Jsang and Roslan Ismail. The beta reputation system. In *Proceedings of the 15th bled electronic commerce conference*, volume 5, pages 2502–2511, 2002.
- [JZF09] Dietmar Jannach, Markus Zanker, and Matthias Fuchs. Constraint-based recommendation in tourism: A multiperspective case study. *Information Technology & Tourism*, 11(2):139–155, 2009.
- [JZFF10] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems*. Cambridge University Press (CUP), 2010.
- [KBV09] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.
- [KFBS14] Thivya Kandappu, Arik Friedman, Roksana Boreli, and Vijay Sivaraman. Privacycanary: Privacy-aware recommenders with adaptive input obfuscation. In *Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on*, pages 453–462. IEEE, 2014.
- [KNN⁺08] Vinod Krishnan, Pradeep Kumar Narayanashetty, Mukesh Nathan, Richard T Davies, and Joseph A Konstan. Who predicts better?: Results from an online study comparing humans and an online recommender system. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 211–218. ACM, 2008.
- [Knu92] Donald E. Knuth. Two notes on notation. *Am. Math. Monthly*, 99(5):403–422, May 1992.
- [LCGM09] Greg Little, Lydia B Chilton, Max Goldman, and Robert C Miller. Turkkit: tools for iterative tasks on mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 29–30. ACM, 2009.
- [LCGM10] Greg Little, Lydia B Chilton, Max Goldman, and Robert C Miller. Turkkit: human computation algorithms on mechanical turk. In *Proceedings of the 23rd annual ACM symposium on User interface software and technology*, pages 57–66. ACM, 2010.

- [LSCP96] David D Lewis, Robert E Schapire, James P Callan, and Ron Papka. Training algorithms for linear text classifiers. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 298–306. ACM, 1996.
- [LVA09] Edith Law and Luis Von Ahn. Input-agreement: a new mechanism for collecting data using human computation games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1197–1206. ACM, 2009.
- [Mar14] Azat Mardan. Redis and authentication patterns. In *Pro Express.js*, pages 171–176. Springer Science + Business Media, 2014.
- [McS03] David McSherry. Similarity and compromise. In *Proceedings of the 5th International Conference on Case-based Reasoning: Research and Development, ICCBR’03*, pages 291–305, Berlin, Heidelberg, 2003. Springer-Verlag.
- [Mit98] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.
- [MS11] Prem Melville and Vikas Sindhwani. Recommender systems. In *Encyclopedia of machine learning*, pages 829–838. Springer, 2011.
- [O’N08] Elizabeth J O’Neil. Object/relational mapping 2008: hibernate and the entity data model (edm). In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1351–1356. ACM, 2008.
- [Paz99] Michael J Pazzani. A framework for collaborative, content-based and demographic filtering. *Artificial Intelligence Review*, 13(5-6):393–408, 1999.
- [PB07a] Rupa Parameswaran and Douglas M Blough. Privacy preserving collaborative filtering using data obfuscation. In *Granular Computing, 2007. GRC 2007. IEEE International Conference on*, pages 380–380. IEEE, 2007.
- [PB07b] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.
- [QB11] Alexander J Quinn and Benjamin B Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1403–1412. ACM, 2011.
- [Ric79] Elaine Rich. User modeling via stereotypes*. *Cognitive science*, 3(4):329–354, 1979.

- [RIS⁺94] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.
- [RMMS04] James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. Dynamic critiquing. In *European Conference on Case-Based Reasoning*, pages 763–777. Springer, 2004.
- [RN05] Francesco Ricci and Quang Nhat Nguyen. Critique-based mobile recommender systems. *OEGAI Journal*, 24(4):1–7, 2005.
- [RUUU12] Anand Rajaraman, Jeffrey D Ullman, Jeffrey David Ullman, and Jeffrey David Ullman. *Mining of massive datasets*, volume 1. Cambridge University Press Cambridge, 2012.
- [RV97] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.
- [Ses10] Peter Sestoft. Numeric performance in c, c# and java. *IT University of CopenhagenDenmark, Version 0.9*, 1, 2010.
- [SFHS] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The Adaptive Web*, pages 291–324. Springer Science + Business Media.
- [SG10] Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In *Recommender Systems Handbook*, pages 257–297. Springer Science + Business Media, oct 2010.
- [SKKR01] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.
- [SSB05] Ellen Spertus, Mehran Sahami, and Orkut Buyukkokten. Evaluating similarity measures: a large-scale study in the orkut social network. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 678–684. ACM, 2005.
- [VG15] Koen Verstrepen and Bart Goethals. Top-n recommendation for shared accounts. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 59–66. ACM, 2015.
- [WG10] Greg Walsh and Jennifer Golbeck. Curator: a game with a purpose for collection recommendation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2079–2082. ACM, 2010.

- [YSC⁺13] Hongzhi Yin, Yizhou Sun, Bin Cui, Zhiting Hu, and Ling Chen. Lcars: A location-content-aware recommender system. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 221–229, New York, NY, USA, 2013. ACM.
- [ZFH⁺08] Markus Zanker, Matthias Fuchs, Wolfram Höpken, Mario Tuta, and Nina Müller. Evaluating recommender systems in tourism—a case study from austria. *Information and communication technologies in tourism 2008*, pages 24–34, 2008.