



Sarah Haas, BSc

Micro-Task Scheduling for Constraint-based Recommendation

MASTER'S THESIS

to achieve the university degree of

Master of Science

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

Abstract

Recommender systems play an important role for online services. As most Internet sites such as Amazon.com or YouTube.com offer many items, it is hard for users to find what they are searching for. Recommender systems try to suggest the best matching items to users based on the available information about the users interests and details about products. These systems are suitable for a wide variety of domains such as movies, music or cars. Recommender systems assist people in dealing with the huge amount of items, the complexity of the item assortment, and also help people with a lack of knowledge in a specific domain to find an appropriate item. Therefore, in a constraint-based recommender system each domain is described by attributes that represent the properties of the available items in detail. These attributes are used in the recommendation process to suggest suitable items to the user. As each user behaves differently and has different interests, she will receive a personalized set of recommendations. The constraint-based recommender developed in this thesis relies on user input to be able to give the best possible suggestions to users. This user input is collected in several ways. One possibility is that users evaluate products. Evaluations in such systems can be time consuming as the users are asked to evaluate several attributes of an item. Users choose the item they want to evaluate and can evaluate one or more attributes. This task can be time consuming and complex as users are confronted with all attributes regarding an item. The evaluations are needed to enable the system to generate suggestions for items. As an alternative way of collecting data, micro-tasks are introduced. Micro-tasks can be solved in a time efficient fashion because users are only asked to declare information about one specific item and attribute. The main advantage is that it takes just a few seconds to solve a micro-task as just one attribute of an item should be evaluated. In this case, the user cannot choose the item as it is automatically assigned. To guarantee a high quality of the information gathered by micro-tasks, it is necessary to find users who have a lot of knowledge in a recommender domain. Therefore, a scheduling algorithm is introduced in this thesis which claims to solve the problem of assigning micro-tasks to users who might be best suited to solve it.

Kurzbeschreibung

Empfehlungssysteme spielen eine wichtige Rolle auf Internet Seiten wie zum Beispiel Amazon.com oder YouTube.com, die eine breite Palette an Produkten oder Dienstleistungen anbieten. Für Benutzer ist es meist sehr schwierig in dieser großen Menge an Möglichkeiten, das für sie passende Produkt oder die passende Dienstleistung zu finden. Empfehlungssysteme unterstützen Benutzer dabei sich in dem riesigen und oft sehr komplexen Sortiment zurechtzufinden sowie beim Finden der passenden Produkte oder Leistungen. Diese Systeme sind auf verschiedenste Arten von Produktgruppen und anderen Leistungsgruppen anwendbar, beispielsweise für die Empfehlung von Filmen, Musik oder Automobilen. Empfehlungssysteme unterstützen Benutzer, die wenig detailliertes Wissen über eine spezielle Produktgruppe besitzen, ein für sie passendes Produkt zu finden. Aus diesem Grund besitzt jede Produktgruppe eigene Attribute mit denen sich die Eigenschaften der Produkte in der jeweiligen Produktgruppe beschreiben lassen. Diese Attribute werden im Empfehlungsprozess benutzt um passende Produkte zu finden. Da jeder Benutzer unterschiedliche Verhaltensweisen und Interessen hat, bekommt er anhand dieser Attribute auf ihn zugeschnittene Empfehlungen. Um dem Benutzer die bestmöglichen Produkte zu empfehlen, sind Daten nötig anhand derer entschieden werden kann, ob ein Produkt für einen Benutzer passend wäre oder nicht. Diese Informationen können auf verschiedene Weisen ins System eingepflegt werden. Eine sehr gängige Art und Weise ist das Erstellen von Evaluierungen für bestimmte Produkte, die der Benutzer gekauft oder in Anspruch genommen hat. Evaluierungen bestehen aus mehreren Fragen bezüglich des Produkts und Benutzer können eine oder mehrere dieser Fragen beantworten. Der Benutzer wählt das zu evaluierende Produkt selbst aus. Da eine Evaluierung allerdings relativ lang dauert, machen sich nicht viele Benutzer die Mühe, Evaluierungen für Produkte zu erstellen. Um trotzdem genügend Informationen zu bekommen wird hier das Konzept der "Micro-Tasks" benutzt. Jeder "Micro-Task" besteht aus einer kurzen Fragen, die der Benutzer innerhalb von weniger als einer Minute beantworten kann. Die Fragen werden automatisch zugewiesen. Da diese Fragen sehr schnell gelöst werden können, tendieren Benutzer eher dazu solche Fragen zu beantworten als langwierige Evaluierungen zu erstellen. Das Schwierige ist nun Benutzer zu finden, die mit hoher Wahrscheinlichkeit die richtige Antwort zu der Frage geben können. Es gilt daher Benutzer zu finden, die sehr viel Wissen in einer bestimmten Produktgruppe besitzen. Um dies zu tun, wird in dieser Arbeit ein Verteilungsalgorithmus entwickelt, welcher versucht für gegebene Micro-Tasks, die Benutzer zu finden, welche am meisten Wissen über dieses Produkt oder die Produktgruppe besitzen.

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.

Date

Signature

Contents

1	Introduction and Motivation	1
2	Related Work	3
2.1	Recommender Systems	3
2.1.1	Collaborative Recommendation	4
2.1.2	Content-Based Recommendation	5
2.1.3	Knowledge-Based Recommendation	6
2.1.4	Hybrid Recommendation	7
2.2	Human Computation	8
2.3	Micro-Tasks	9
2.4	Games	11
2.5	Similarities and improvements to related work	12
3	Basic Recommendation Approach	13
3.1	Definitions	13
3.2	Recommendation Approach	20
3.2.1	Result Set	20
3.2.2	Support of Item Attributes	21
3.2.3	Aggregated Support	22
3.2.4	Utility Function	23
4	PeopleViews System Description	26
4.1	PEOPLEVIEWS Architecture	26
4.2	User Interfaces	29
4.3	Recommender Interfaces	30
4.4	Item Interfaces	33
4.5	Micro-Task Interfaces	36
4.6	Recommendation Interfaces	41
4.7	Game	43
5	Micro-Task Scheduling Approach	47
5.1	TF-IDF	48
5.1.1	TF	48
5.1.2	IDF	49

5.2	Extraction and Weighting of Keywords	49
5.2.1	Keyword Extraction for Items	49
5.2.2	Keyword Extraction for Users	51
5.2.3	Weighting of Item Keywords	52
5.2.4	Weighting of User Keywords	53
5.2.5	Extraction and Weighting of Recommender Keywords	55
5.2.6	Extraction and Weighting of Micro-Task Keywords	55
5.3	Agenda for Micro-Task Generation	55
5.4	Scheduling Approach	61
5.4.1	Definitions	61
5.4.2	Example	65
6	Evaluation	75
6.1	Dataset	75
6.2	Evaluation Approach	78
6.3	Evaluation Results	84
7	Limitations and Future Work	89
7.1	Limitations	89
7.1.1	Technical Limitations	89
7.1.2	Limitations of the Scheduling Algorithm	90
7.2	Limitations of the Evaluation	90
7.3	Future Work	90
8	Conclusion	92

1

Introduction and Motivation

Recommender systems are used to support users in finding suitable products within a large amount of items. Recommenders, nowadays, are integrated in many online services and utilized by users to make decisions such as which item to purchase or which movie to watch.

In this thesis, a constraint-based multi-domain recommender system was implemented which tackles the problem of finding the best possible recommendations for users. Furthermore, the acquisition of data to be able to generate the recommendations is a main aspect of this thesis. Data acquisition is an essential part of constraint-based recommender systems as they need information about items to be able to provide appropriate recommendations. Multi-domain, in this case, means that the system can deal with several recommender domains such as skiing resorts, mobile phones or cities within the same system. Although, those recommender domains have nothing in common, their different attributes can be modeled in PEOPLEVIEWS ¹.

In contrast to approaches such as collaborative filtering, constraint-based recommenders can deal with very complex items. To add information to the system, it is necessary that users evaluate items. For example, such information could be the target audience of a skiing resort or the performance of a mobile phone. In order to make data acquisition as efficient as possible, one of the goals of this thesis is to find an appropriate way to acquire preference information regarding items. One possibility to acquire information in PEOPLEVIEWS are evaluations where users actively select an item and answer one or multiple questions regarding the item's attributes.

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

Another way of acquiring data are *micro-tasks*. Micro-tasks contain one single question about a specific item's attribute with one or more possible answers. These tasks can be solved in a minimum of time which should encourage users to contribute to the system. As the attributes of items can have different characteristics such as single answer or multiple answers, different types of micro-tasks are proposed in this thesis. Micro-tasks, in contrast to evaluations, are assigned to users by the system. The differences between evaluations and micro-tasks will be described in detail in Section 3.1. Thus, PEOPLEVIEWS needs to find appropriate users which the system assumes to be best suited to complete a specific micro-task. Best suited users are users with properties such as having knowledge about the item or recommender domain, having answered many micro-tasks before, or having a low workload. Finding such users leads to the main goal of the thesis which aims to solve the selection of suitable users to solve micro-tasks. To deal with this issue, an approach is proposed that claims to be able to find the best matching user for a micro-task based on the user's past interactions with the system.

This thesis is organized as follows: Chapter 2 contains a discussion of related work in recommender systems, micro-tasks, and scheduling approaches. In Chapter 3 the PEOPLEVIEWS recommendation approach, as well as definitions of terms and concepts used in this thesis are introduced. Furthermore, two example recommender applications are constructed from scratch to be able to provide meaningful examples throughout the thesis. Chapter 4 contains the description of all aspects of the PEOPLEVIEWS user interface, the description of a game with a purpose and also the explanation of the technical realization of the user interface as well as the PEOPLEVIEWS server. Games with a purpose are used to gather information. In this case, the goal is to answer questions regarding items of a chosen recommender domain. Such games can be used to motivate people to contribute to the system. The different types of micro-tasks will also be shown in Chapter 4. In Chapter 5, the scheduling of micro-tasks is described. Furthermore, the extraction and weighting of keywords extracted from data sources such as item descriptions are explained. The keywords are needed to help in deciding if a user is capable of completing a specific micro-task. The concept of an agenda and the construction of micro-tasks from such an agenda are discussed. Agendas are provided by the PEOPLEVIEWS Quality Assurance component and indicate which micro-tasks need to be generated in order to acquire enough information about specific items to improve the recommendation quality. The evaluation of the scheduling algorithm as well as the baseline algorithms and the used data set are described in Chapter 6. Chapter 7 includes a discussion of the limitations of the scheduling algorithm, the evaluation data set, the evaluation itself, and also the technical limitations of the system. Furthermore, open issues for future work are discussed. The thesis is concluded with Chapter 8.

2

Related Work

In this chapter, an overview of related work is provided. First, different kinds of recommender systems will be discussed in Section 2.1. Section 2.2 analyzes the related work in the context of human computation. Section 2.3, provides an overview of approaches that explain the concept of micro-tasks. Section 2.4 covers aspects regarding games with a purpose.

2.1 Recommender Systems

Resnick and Varian [1] state that users often have to make decisions without having sufficient knowledge needed for taking the decision properly. They suggest to use recommender systems to assist people in making good decisions and give a definition of the term *recommender systems*.

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.

The authors also state that recommender systems should assist and augment the natural social process of relying on recommendations from other people by word of mouth, reviews in newspapers, etc.

One recommender system was developed by Goldberg et al. [2], who introduced *Tapestry*, a mail system which uses collaborative filtering to select relevant documents. As content-based filtering did not solve the problem of receiving unwanted

mails properly, they proposed a *collaborative filtering approach*. Collaborative filtering is implemented in terms of a collaboration between users who help others to perform filtering by annotating documents they read.

In their paper, Felfernig and Burke [3] give an overview of current state-of-the-art recommender systems. They categorize them into four different classes: collaborative, content-based, knowledge-based, and hybrid recommendation approaches. Generally, these categories are defined as follows:

- Collaborative recommendation: Recommends items that other users with similar preferences liked in the past.
- Content-based recommendation: A user gets recommendations for items that have commonalities to the items she preferred in the past.
- Knowledge-based recommendation: Recommendations are generated by reasoning about which items might be relevant for the user by using domain knowledge (represented, e.g. in the form of constraints) and user requirements.
- Hybrid recommendation: Recommendation approaches are combined in such a way that the weaknesses of one approach are compensated by the other one.

These four categories will be described in more detail in the next subsections.

2.1.1 Collaborative Recommendation

Breese et al. [4] analyze different algorithms for collaborative filtering. They also give a definition of collaborative filtering.

The task in collaborative filtering is to predict the utility of items to a particular user (the active user) based on a database of user votes from a sample or population of other users (the user database).

They also distinguish between algorithms that *always* take the entire user database to make predictions and algorithms which use the database *once* to learn models which are later used to predict items. The results show that the first type of algorithms is slower and needs more memory resources. The other type, however, needs less memory but requires a learning phase.

Konstan et al. [5] applied collaborative filtering to Usenet news. This system includes a newsgroup with hundreds of messages per day. Due to the huge amount of information, they integrated collaborative filtering to recommend relevant articles to user. Each user had to rate several articles before he could receive recommendations. This was considered a problem because many users did not want to make this

contribution. This indicates that collaborative filtering has problems with cold-start [6] where there is no or very little user data available.

In their paper, Sarwar et al. [7] introduce an item-based collaborative filtering approach. Their approach avoids the bottleneck of traditional collaborative filtering approaches which arises when searching for potential neighbors in a large set of possible neighbors. To avoid this, they first explore the relations between items and not between users. The results show that an item-item-based matching approach achieves a better prediction quality than conventional user-user-based matching. A very popular website which uses item-based collaborative filtering is Amazon.com. Linden et al. [8] describe the approach and state that it can be used for efficient marketing and also scales for a large retailer such as Amazon.com.

2.1.2 Content-Based Recommendation

Pazzani et al. [9] give a definition of content-based recommender systems as

[...] systems that recommend an item to a user based upon a description of the item and a profile of the user's interests.

The authors discuss different item representations from which the necessary information for recommending items can be extracted. They first describe a data structure in a database, which already contains arbitrary items of a single domain such as books. For each item, the item details are already stored in the database. Item details, for example, are price or author of a book. As it is not always possible to get information in such a structured way, another approach described by the authors tries to extract information from unstructured data. They use the TF*IDF technique [10] for unstructured text to extract words and weight them. In their case, item descriptions are a form of unstructured text.

TF*IDF is a technique that calculates the importance of a word in a collection of documents. For each word a weighting value between 0 and 1 is calculated that indicates if the word is more or less important in this collection of documents. The weighting values for each word of an item description can be calculated with this technique indicating the importance of a certain word of the description. TF*IDF will be further described in Section 5.1 as it will be used in the task scheduling approach introduced in this thesis to create a history of users' interests stored in so-called user profiles.

Pazzani et al. [9] also introduce user profiles which hold the user's preferences such as types of items she is interested in and also a history of the user's interactions with the system. They store keywords weighted using TF*IDF in the user profile. This is similar to the user profile that will be used in this thesis.

They also list limitations such as the need for users to interact with the system and the difficulty of finding enough users who contribute to the system. Furthermore, Pazzani et al. [9] match items based on the extracted keywords in the user

profile and state that currently there are no techniques that create a ranking of the recommended items.

The authors also discuss several techniques of recommendation algorithms such as decision trees, nearest neighbor methods or probabilistic text classifiers to create user profiles used for recommending items. They give an example of how to match the weighted words in the user profile to items by summing up the weights of all words from the user profile that match words of the item description. Furthermore, Pazzani et al. [9] state that the quality of content-based recommendation strongly relies on the quality and quantity of available information such as item descriptions.

Van Meteren and van Someren [11] use the relevance feedback method which is similar to TF*IDF to generate a user profile. This method represents documents and profiles as vectors. In their approach, the profile only consists of one vector containing the topics of interest. The user profile vector is always updated after a user read a document for a certain amount of time. The profile is updated by applying a weighting factor to the previous profile vector plus adding a document vector weighted by the relative importance of the document to the user. The document vector represents the documents read by the user. For example, if a user read several documents regarding the same topic, the topic is weighted higher in the user profile vector.

Furthermore, the authors match the document vector and the user profile to generate recommendations. The results of the paper show that content-based filtering approaches cannot predict the user's future interest which would be possible with collaborative filtering approaches. They state that it might be more effective to combine collaborative and content-based filtering approaches to achieve better results. However, the authors do not further elaborate on that idea.

2.1.3 Knowledge-Based Recommendation

Burke [12] defines knowledge-based recommendation as an approach that generates a recommendation by reasoning about which products users might be interested in. This is done by using knowledge about users and products. The author also points out that knowledge-based recommendation avoids some of the drawbacks of the other approaches. For example, the cold-start problem does not occur as this approach does not rely on user ratings to generate recommendations.

Felfernig et al. [13] define knowledge-based recommendation as consisting of a recommendation task and a recommendation. The recommendation task is defined as a constraint satisfaction problem [14]. It consists of a set of domain variables describing product properties, constraints describing user's requirements, filter constraints, and a constraint specifying the set of available products.

Recommendation itself is defined as follows: An assignment of variables is a recommendation if each of the individual assignments is consistent with the constraints

of user requirements, filter constraints, and the constraints of the set of available products.

For the system implemented in this thesis, a special form of knowledge-based recommendation is used, namely constraint-based recommendation. Felfernig and Burke [3] give the following definition:

In this paradigm recommendation is viewed as a process of constraint satisfaction, some constraints come from users, other constraints come from the product domain. Products that satisfy the constraints are good recommendations.

Products that satisfy the constraints are suitable recommendations. They also state that this kind of recommender system does not encounter a problem with cold start and is also able to recommend products that are not used enough to generate a meaningful history (for example, a reading history). Felfernig et al. [15] showed that such recommender systems can be used in the financial sector too.

Felfernig et al. [16] recently wrote a paper regarding multi-domain recommender systems using constraint-based recommendation and human computation. This method can be used to host multiple recommender domains in one system. They also show that users are willing to contribute to the system but don't want to invest a large amount of time. Felfernig et al. [17] discussed recommendation approaches in the context of multi-domain recommender systems and furthermore, parts of the PEOPLEVIEWS user interface are described.

2.1.4 Hybrid Recommendation

Robin Burke [18] discusses hybrid recommendation approaches such as a combination of knowledge-based recommendation and collaborative filtering techniques. He also gives a definition of hybrid recommendation approaches.

Hybrid recommender systems combine two or more recommendation techniques to gain better performance with fewer of the drawbacks of any individual one. Most commonly, collaborative filtering is combined with some other technique in an attempt to avoid the ramp-up problem.

Burke, furthermore, discusses the advantages and disadvantages of different combinations of approaches. He states that some combinations are able to reduce or even remove the disadvantages of a single approach. For example, when combining a knowledge-based with a collaborative filtering approach the cold-start problem can be mitigated.

Burke [19] discusses hybrid web recommender systems. He addresses two-part hybrid recommender systems. It turns out that *cascade hybrids* are generating the best results in terms of recommendation quality [20]. Such approaches create a hierarchical structure consisting of two stages as two different recommendation approaches are used. If more recommendation approaches would be used, the structure would consist of more stages. In the first stage, the items are preprocessed with a recommendation approach such as knowledge-based recommendation that, for example, removes items that will definitely not match. In the second stage, the items left are ordered using some other recommendation technique, for example, collaborative filtering.

Several researchers used hybrid approaches to implement applications. Albadvi and Shahbazi [21] combined collaborative and content-based filtering approaches to recommend items based on product category preferences of users. They track the buying behavior and purchases of the user. They use the bought products to filter possible items using collaborative filtering and then rank the remaining products using the product details with a content-based approach. Lekakos et al. [22] also used a combination of collaborative and content-based filtering to recommend movies. They use a similar approach as Albadvi and Shahbazi [21] used. First, the movies a user watched were used to select possible movies using collaborative-filtering. Afterwards, they rank the movies left with a content-based approach using the movie descriptions to extract keywords and match the extracted keywords to find appropriate movies.

2.2 Human Computation

Von Ahn [23] describes human computation problems as problems which can be solved by humans easily but are hard to solve for computers.

By leveraging human abilities in a novel way, I solve large-scale computational problems and collect data to teach computers basic human talents.

As an example, he states that CAPTCHAs containing text snippets which were used for digitalization of words contained in a book can be displayed in squiggly characters. Digitalization of books is a hard task for computers because recognize squiggly characters is very challenging for computers. CAPTCHAs are typically entered by humans for whom this is an easy task. The concept is to distribute massive computational problems to humans to solve it.

Quinn and Bederson [24] try to give a more formal definition of human computation problems based on the work of von Ahn's doctoral thesis [25] which inspired the term "human computation".

The problems fit the general paradigm of computation, and as such might someday be solvable by computers. The human participation is directed by the computational system or process.

The authors also state the difference between human computation and crowdsourcing, social computing, collective intelligence or data mining.

- Human computation: Using humans abilities to solve tasks that are hard for computers but easy for humans.
- Crowdsourcing: Jobs usually done by traditional human workers are outsourced to a large group of members of the public.
- Social computing: Natural human behavior is mediated by technology. The purpose is not to perform a computation.
- Data mining: Extraction of patterns from data using specific algorithms.
- Collective intelligence: Combination of the four previous concepts in which groups of humans complete tasks collectively that appear to be intelligent.

Yuen et al. [26] provide a categorization of existing human computation systems. They differentiate between initiatory human computation, distributed human computation, and social game-based human computation. Initiatory human computation means to collect commonsense knowledge by letting users interact with the system. Distributed human computation addresses the huge number of users on the Internet who contribute to the system. Social game-based human computation means to use social games to solve difficult AI problems with the help of users. Social games are entertaining for users and while playing games they provide information to the system in an enjoyable matter.

2.3 Micro-Tasks

In this section, existing research regarding the scheduling and distribution of micro-tasks to users is discussed. In this thesis, micro-tasks are related to the context of human computation, therefore, an overview of related work regarding this context is given.

Micro-tasks are defined by Sarasua et al. [27] as

[...] a problem is outsourced to a distributed group of people by splitting the problem space into smaller sub-problems, or tasks, that multiple workers address independently [...]

Schnitzer et al. [28] present a study that focuses on the evaluation of micro-worker preferences in terms of recommendations of tasks in crowdsourcing platforms. The result shows that most workers just want tasks that minimize the amount of time and maximize the amount of money as it would be expected. But workers are also very interested in simplicity and similarity to previously solved tasks. The paper shows that in order to achieve results of higher quality, scheduling algorithms should not only take into account the value for money when distributing tasks but also preferences of the workers.

Ambati et al. [29] analyzed the work flow of users choosing random micro-tasks to solve on Amazon Mechanical Turk ². The interface of the Amazon Mechanical Turk shows random tasks and does not adapt on behalf of a current worker's preferences and skills. The authors discovered that this representation is not optimal as not every worker is able to solve every task. This leads to a low quality of the solved tasks as less-skilled workers solve tasks to earn a high amount of money even if they would need much better skills. On the other hand, better-skilled workers need to search rather long to find appropriate tasks as also easy to solve tasks are shown to them. To overcome this problem, Ambati et al. [29] proposed two methods for building user preference models which can then be matched with the micro-tasks. Both approaches rely on keywords which can be extracted from items. The interactions of users are stored in a database and can later be used to generate keywords and match them to keywords extracted from micro-tasks.

Hadano et al. [30] propose a method to solve the task assignment problem in mobile crowdsourcing platforms. They use worker-task graphs and add time constraints to the graphs. Each task has a deadline to complete the task. Hadano et al. analyzed the time workers invested in tasks to solve them in past periods. Using this data, the authors then checked for all tasks if workers are available to solve the tasks in time. When adding the time constraints a lot of connections between workers and tasks are removed due to not being able to solve the task within time. This leads to a remaining set of tasks which is recommended to a worker and also to an increase of completed tasks per time unit in this system.

Rajan et al. [31] propose the system CROWDCONTROL, which is used to control the scheduling of large batches of tasks to workers. Instead of distributing all tasks at once, the set of tasks is divided into small subsets which are assigned to workers in several rounds. After every round, the performance of the workers, as well as the fulfillment of the requirements (such as the minimum number of solved tasks in a time unit) specified by the task creator, are checked. The performance in the previous round influences the setup of the next round. If, for example, workers were able to solve very complex tasks, they will receive complex tasks again in the next

² <https://www.mturk.com/mturk/welcome>

round. On the other hand, if workers needed very long to solve the tasks, they will get easier tasks that can be solved faster.

Wang [32] addresses the problem of task scheduling based on user satisfaction. He states that most algorithms use the same satisfaction function for all users and assume that each user has the same motivation to solve tasks. He proposes an approach where each user has its own satisfaction function and the overall satisfaction can be computed from the satisfaction functions of the users. He uses a genetic algorithm to improve the weighting parameters influencing the timing and budgeted constraints used in the satisfaction function of the user. His study shows that the overall user satisfaction increases with the algorithm he invented as the workers get the amount of tasks they are willing to solve.

2.4 Games

The PEOPLEVIEWS system contains a game with a purpose which is used to get further information about users and products as well as to motivate users to contribute to the system. Deterding et al. [33] address the usage of game-design elements in non-gaming contexts and state that

The most recent phenomenon in this trajectory is ‘gamification’, an umbrella term for the use of video game elements (rather than full-fledged games) to improve user experience and user engagement in non-game services and applications.

Zichermann and Cunningham [34] describe all aspects that need to be considered regarding gamification in web applications. They point out that gamification is a huge motivation factor for users to contribute to a system. They state that the quality of data produced by games is very good because people try to maximize their score. Furthermore, they show mechanisms such as badges and points users can earn by playing which makes users really feel like playing a game rather than just solving tasks.

Hamari et al. [35] examine if gamification actually works and which advantages it has. They review existing empirical research in gamification to identify the impact of gamification on different kinds of systems. The results show that gamification actually has positive impacts on the frequency of system usage, but they also show that it strongly depends on the context in which the gamification is implemented. The studies showed that users were more motivated to use a system when games were involved especially in the context of educational and learning systems. They also showed that in systems such as e-commerce platforms it is challenging to add gamification as in such systems users are more interested in maximizing their profit than having fun.

2.5 Similarities and improvements to related work

The recommender system PEOPLEVIEWS is based on a constraint-based recommendation approach. In contrast to other knowledge-based approaches PEOPLEVIEWS allows to freely add and configure arbitrary recommender domains.

The micro-task scheduling approach implemented in this thesis is based on a content-based approach. In addition to item descriptions which are used by most approaches also other factors such as excellence or workload of a user and several other metrics which are described in detail in Section 5 influence the micro-task scheduling.

The concept of human computation is applied in PEOPLEVIEWS by introducing techniques and algorithms that aggregate user input which can then be used to generate recommendations. The micro-task approach implemented in this thesis increases the applicability of human computation in constraint-based recommenders.

Gamification and awarding points for interaction with the system such as adding items is used to motivate users to contribute to the system. The game with a purpose implemented in PEOPLEVIEWS should motivate users to increase their knowledge about the domains to maximize their highscore.

3

Basic Recommendation Approach

The system implemented within the scope of this thesis is based on constraint-based recommendation. The general definition of a constraint-based recommender was already given in Chapter 2. As this system should be able to deal with the creation and maintenance of a large number of recommenders, the initial definition of a constraint-based recommender needs to be adapted and extended. To provide a better understanding of the concepts described in the following section, two example recommenders will be constructed. These recommenders will be used throughout the entire remaining thesis. Furthermore, the recommendation approach used by the system to generate recommendations for users will be described in this chapter.

3.1 Definitions

User $u \in U$. A user u can contribute to the systems by solving micro tasks, playing games, creating evaluations, etc. But as the system supports both, contributing and non-contributing users, a user u could also just use the recommender to get recommendations. Contributing and non-contributing users are not distinguished by the system and treated in the same way. Each user is allowed to add *items* to *recommenders*. In Table 3.1 an example of users in the system is shown.

Recommender $r \in R$. A recommender is related to one specific domain. A recommender consists of name, description, image, the *item attribute* and *user attribute* definitions. With these definitions it provides a skeleton for *items*

Username	E-Mail
Mike	mike@test.com
James	james@test.com
Linda	linda@test.com
Marry	marry@test.com

Table 3.1: Examples of PEOPLEVIEWS users.

that can be added to the recommender. Each recommender consists of several *items*, each *item* added to the same recommender consist of the same *item attributes* and *user attributes*. As mentioned before, the creator of the recommender defines the *item attributes* (hard or objective facts about items) and *user attributes* (soft or subjective facts about items) for the recommender. Table 3.2 shows examples of recommenders.

Recommender Name	Description
Mobile Phones	Want to buy a new mobile phone? This recommender will help you make a decision.
Skiing Resorts	You like skiing or snowboarding? You love the winter and the mountains? We help you find the perfect resort.

Table 3.2: Examples of recommenders in the system.

Item Attributes $ia \in IA$. These describe objective properties of *items*. Item attributes are *hard facts* of an item such as the display size of a smart phone. These attributes are of different type such as numerical, textual or predefined values from which the item creator can choose. The type of each item attribute is defined by the creator of the recommender. The domain values of the item attributes are entered by the *user* who adds an *item* to the system; the item attributes themselves are entered by the creator of the recommender. Therefore, the possible item attributes are the same for each *item* in a specific *recommender*, only the values are different. Each item attribute also needs an order relation. In PEOPLEVIEWS, the following order relations or similarity metrics for numbers can be used: *less is better*(LIB), *more is better*(MIB), *nearer is better*(NIB) or *equal is better*(EIB) [36]. The order relations of item attributes are used in the recommendation process to allow the system to rank items based on item attributes. The order relation for each item attribute is defined by the creator of the recommender. For text input and predefined answers (enumerations), only *Exact match*(EM) and *Display only*(DO) can be chosen, only these two similarity measures can directly be applied to text. *Exact match* is treated the same way as *equal is better* is. *Display only* means

that the item attribute is only shown to the user as information but cannot be used to match with user-defined filter criteria. Table 3.3 gives some examples of item attributes.

Item Attribute	Input Type	Enumeration Answers	Order Relation	Recommender
OS	enumeration	{Blackberry, iOS, Android, Windows}	EIB	Mobile Phones
Release Date	text		DO	Mobile Phones
Display Size	number		EIB	Mobile Phones
Memory	number		MIB	Mobile Phones
Price	number		LIB	Mobile Phones
Family friendly	enumeration	{Yes, No}	EM	Skiing Resorts
Sledging	enumeration	{Yes, No}	EM	Skiing Resorts
Cross-country skiing	enumeration	{Yes, No}	EM	Skiing Resorts
Kilometers of slopes	number		MIB	Skiing Resorts
Types of lift facilities	text		DO	Skiing Resorts

Table 3.3: Examples of item attributes for the mobile phones and skiing resorts recommenders can be seen. The different input types and similarity measures are shown as well.

User Attributes $ua \in UA$. In contrast to *item attributes*, user attributes are subject to the users opinion. They can be seen as *soft facts* about items, such as the usability of a smart phone. For instance, *user* u_1 could consider the usability for smart phone *Samsung Galaxy S7* as *very good*, whereas *user* u_2 could consider it as *moderate*. The answers to the user attributes are given by *users* who evaluate an already existing *item*. So neither the item creator nor the creator of the recommender enter those values but evaluating *users* do. Another difference to *item attributes* is that the answers of user attributes are predefined. The evaluating *user* has to choose one or more possible answers and cannot enter it as, for example, textual input. These answers are called *user attribute values* $uav \in UAV$. For instance, the usability of a smart phone may have the following possible user attribute values: *very good*, *good*, *moderate*, *bad* and *very bad*. There are two different types of user attributes: single-answer and multi-answer. An example of a multi-answer attribute is, for example, the *target audience* for a skiing resort with the user attribute values *family*, *recreational athlete*, *top-class athlete* or *snowboarder*. There it makes

sense to choose more than one user attribute value as the skiing resort might have more than one possible *target audience*. The user attributes as well as the *user attribute values* are defined by the creator of the recommender. Examples of user attributes and user attribute values are shown in Table 3.4. For each user attribute, a question which is asked to users who evaluate an existing item needs to be specified as well. Because of space limitations, this text is omitted in the examples shown in Table 3.4.

User Attribute	User Attribute Values	Choice Type	Recommender
Performance	{poor, acceptable, good, excellent}	single	Mobile Phones
Value for money	{keeps value, loses value slightly, loses value fast}	single	Mobile Phones
Battery life for different usages	{photo, gaming, browsing, telephoning}	multiple	Mobile Phones
Design	{valuable, moderate, poor}	single	Mobile Phones
Price level	{expensive, moderate, cheap}	single	Skiing Resorts
Experience level needed	{ high, moderate, low }	single	Skiing Resorts
Quality of lift facilities	{new, average, old}	single	Skiing Resorts
Target audience	{family, recreational athlete, top-class athlete } snowboarder }	multiple	Skiing Resorts
Quality of ski lodges	{good, moderate, bad}	single	Skiing Resorts

Table 3.4: Examples of user attributes and corresponding user attribute values for mobile phone and skiing resort recommenders.

Item $i \in I$. Items are the entities which get recommended by the recommender system. Items consist of *name*, *description*, *image*, *tags*, *item attributes*, and the *recommender* they belong to. When creating a new item, the values of the *item attributes* are added by the creator of the item. After creation, each item can be evaluated by users. Users specify how well they think the *item* supports certain *user attribute values* (see *support*). In the Tables 3.5 and 3.6 examples of items in both recommenders from Table 3.2 can be seen. To keep the tables simpler, the item descriptions and tags are omitted.

Item Name	OS	Release Date	Display Size	Memory (in GB)	Price
Samsung Galaxy S7	Android	February 2016	5,1	32	588
Apple iPhone 6S	iOS	September 2015	4,7	16	610
Sony Xperia M5	Android	December 2015	5,0	16	307

Table 3.5: Example items of the recommender Mobile Phones.

Item Name	Family Friendly	Sledging	Cross-country skiing	Kilometers of slopes	types of lift facilities
Kitzbühel	Yes	Yes	Yes	173	cableway, chair lift, drag lift
Schladming	Yes	No	Yes	123	cableway, chair lift, drag lift
Obertauern	Yes	Yes	Yes	100	cableway, chair lift, drag lift

Table 3.6: Example items of the recommender Skiing Resorts.

Support $S_{uav} \in S$. The support is needed when an *item* is evaluated. It defines how well the different *user attribute values* for each *user attribute* are supported. The support is a percentage value between -1 and 100% . -1 indicates that the *user* has no knowledge about the support for this specific *user attribute value*. If the user chooses -1 , the system will not use the support specified for this *user attribute value* when calculating recommendations. The percentage value assigned by users to item-user attribute value combinations allows the system to distinguish how strongly this *user attribute value* is supported by the evaluated *item*. For instance, the support for the user attribute *usability* for the *item Samsung Galaxy S7* could be 70% for being *very good* whereas for the *item Apple iPhone 6S* it could be 90% . This means that both smart phones have very good usability, but the *Apple iPhone 6S* is still better than the *Samsung Galaxy S7* when it comes to usability. The support can be entered by an evaluating *user* for each *user attribute* and at least one *user attribute value*. The support is needed in the calculation of the recommendation of *items* to a *user*. For each *item-user attribute value* combination, there exists exactly one support value. The Tables 3.7 and 3.8 contain the evaluations of different items from different users. The cells contain the support and *user attribute value* the user defined for a certain *user attribute*.

User	Item	Performance	Value for money	Battery life for different usages	Design
Mike	Samsung Galaxy S7	acceptable (80%)	looses slightly (60%)	gaming (80%) browsing (75%) photo (70%)	valuable (80%)
James	Samsung Galaxy S7	very good (75%)	looses slightly (80%)	photo (70%) gaming (85%)	moderate (90%)
Linda	Apple iPhone 6S	very good (90%)	keeps value (70%)	photo (80%) browsing (75%)	valuable (80%)
James	Sony Xperia M5	acceptable (60%)	looses fast (60%)	telephoning (80%) browsing (70%)	moderate (70%)
Marry	Sony Xperia M5	very good (75%)	looses slightly (80%)	browsing (85%) photo (75%)	valuable (90%)
Mike	Apple iPhone 6S	excellent (80%)	looses slightly (75%)	gaming (80%) browsing (70%) photo (60%)	moderate (80%)

Table 3.7: Example evaluations of items of the recommender Mobile Phones. The support is given as a percentage value in brackets.

Requirements $REQ_{uav}, REQ_{ia} \in REQ$. Requirements define a set of *user attribute values* and *item attribute values* selected by the user. The *user* specifies those criteria to get recommendations. The recommendation algorithm then tries to find the best matching *items* for the specified criteria. There is a difference between requirements for *user attribute values* REQ_{uav} and *item attribute values* REQ_{ia} as the support for those is calculated differently which will be shown in Section 3.2.

User Profile $p_u \in PU$. A user profile consists of weighted keywords. Those keywords are extracted by analyzing the *user's* interactions with the system. A detailed description of the keyword extraction will be given in Chapter 5. The keywords are later used to match *users* and *items* of *micro-tasks* when scheduling the *micro-tasks*. Examples of user profiles will be given in Chapter 5, as the keyword extraction is shown.

Item Profile $p_i \in PI$. An item profile is similar to a *user profile*, but only consists of weighted keywords that were extracted from the item description and item tags. Item profiles are also used to match *users* and *items* of *micro-tasks* when *micro-tasks* are scheduled. How the extraction and weighting are done, will be described in detail in Chapter 5 and also example item profiles will be shown.

User	Item	Price level	Experience level needed	Quality of lift facilities	Target audience	Quality of ski lodges
Marry	Kitzbühel	exp. (80%)	moderate (90%)	average (70%)	rec. athlete (70%) snowboarder (80%)	good (70%)
James	Kitzbühel	mod. (90%)	high (75%)	new (80%)	top. athlete (70%) rec. athlete (70%)	moderate (80%)
James	Schladming	mod. (80%)	moderate (75%)	new (70%)	rec. athlete (80%) rec. family (90%)	good (80%)
Linda	Schladming	mod. (70%)	low (70%)	average (90%)	snowboarder (80%) family (80%)	good (90%)
Marry	Obertauern	cheap (70%)	moderate (65%)	new (80%)	rec. athlete (70%) family (65%)	moderate (75%)

Table 3.8: Example evaluations of items of the recommender Skiing Resorts. The support is given as a percentage value in brackets.

Micro-Task $mt \in MT$. Micro-tasks are used to acquire supports for *item - user attribute value* combinations, similar to evaluations. The differences between those two concepts are:

- When evaluating an item, users self-initiated select which item to evaluate, micro-tasks get assigned to users by the system and ask them to specify supports for an automatically chosen item.
- Evaluations allow users to specify supports for all defined *user attribute values*, whereas micro-tasks ask users to specify the support for an automatically selected *user attribute value* or at most for all *user attribute values* of a single *user attribute*.
- Evaluating an item might seem more complex and requires more time than answering a single micro-task. However, by evaluating an item, of course more information can be specified than by computing a single micro-task.

Each user gets several micro-tasks assigned which she should solve. The purpose of a micro-task is to acquire *support values* of *user attribute values* for different *items*. As already stated in Chapter 2, *users* do not want to invest a lot of time into maintaining a recommenders knowledge base [16].

There are different micro-task types in the system as there are different types of user attributes. The purposes and different types of micro-tasks will be further described in Chapter 4. The assignment of micro-tasks to users is, among other things, based on the *user profiles* and the *item profiles* and will be discussed in detail in Chapter 5. The generation of micro-tasks will also be described in Chapter 5.

3.2 Recommendation Approach

One main functionality of PEOPLEVIEWS is the generation of recommendations for users. In this section, the PEOPLEVIEWS recommendation approach will be presented.

3.2.1 Result Set

Equation 3.1 gives the final result set RS, which represents the items that are shown to the user as a recommendation based on the user's chosen requirements. The requirements are the filter criteria provided by the user. $REQ_{uav} = \emptyset$ and $REQ_{ia} = \emptyset$ indicate that for these user attribute values and item attributes no requirements were defined by the user. For example, a user could define the requirements shown in Table 3.9 to get recommendations from the Skiing Resorts recommender. Table 3.10 shows the result set for the criteria given in Table 3.9.

Type	Requirement	Value
User Attribute	Price Level	moderate
User Attribute	Target Audience	family, snowboarder
Item Attribute	Family friendly	Yes
Item Attribute	Kilometers of slopes	120

Table 3.9: Example of requirements a user could define for Skiing Resorts.

$$RS = \{i \mid \forall uav \in REQ_{uav} : \text{support}_{\Sigma}(i, uav) > 0 \vee REQ_{uav} = \emptyset\} \cap \{i \mid \forall ia \in REQ_{ia} : \text{support}_{\Sigma}(i, ia) > 0 \vee REQ_{ia} = \emptyset\} \quad (3.1)$$

Item
Kitzbüchel
Obertauern
Schladming

Table 3.10: Result set RS for the requirements given in Table 3.9.

3.2.2 Support of Item Attributes

The previously mentioned similarity measure NIB, MIB, LIB where defined by McSherry [36]. EIB is not mentioned in this paper as it is just a special case of NIB. The following equation shows the computation of the support for an item attribute of a specific item for different similarity measures:

$$\text{support}_{\Sigma}(i, ia, iav) = \begin{cases} [iav = \text{val}(i, ia)] & \text{EIB} \\ 1 - \frac{|iav - \text{val}(i, ia)|}{\max(I, ia) - \min(I, ia)} & \text{NIB} \\ \frac{\text{val}(i, ia) - \min(I, ia)}{\max(I, ia) - \min(I, ia)} & \text{MIB} \\ \frac{\max(I, ia) - \text{val}(i, ia)}{\max(I, ia) - \min(I, ia)} & \text{LIB} \end{cases} \quad (3.2)$$

For the example requirements in Table 3.9, the item attributes *Family Friendly* and *Kilometers of slopes* were selected. The computation of the support for both will be shown in the next equations. To do so, the corresponding similarity measure equation for each item attribute needs to be chosen. The calculated supports for the items can be seen in Table 3.11.

For the item attribute *Family Friendly*, the similarity measure is *Exact Match*(EM) which is treated as *equal is better*(EIB) as already stated in Section 3.1. The support for the EIB measure can only take two values, either 100% or 0% as can easily be seen in Equation 3.2. The required value can either be equal to the one given in the item or not. As an example for *Family Friendly* and *Kitzbüchel* is calculated in the following equation:

$$\text{support}_{\Sigma}(\text{Kitzbüchel}, \text{FamilyFriendly}, \text{Yes}) = 100\% \quad (3.3)$$

For the item attribute *Kilometers of Slopes*(KoS), the similarity measure is *more is better*(MIB). To calculate the support, we need to find the largest and smallest value for *Kilometers of Slopes* in the recommender *Skiing Resorts*. The largest value $\max(I, \text{KoS})$ is 173 from item *Kitzbüchel* and the smallest value $\min(I, \text{KoS})$ is 100 from the item *Obertauern*. As an example, the support for item *Schladming*(Schl)

is calculated as follows:

$$\begin{aligned} \text{support}_{\Sigma}(\text{Schl}, \text{KoS}, 123) &= \frac{\text{val}(\text{Schl}, \text{KoS}) - \text{min}(\text{Schl}, \text{KoS})}{\text{max}(\text{Schl}, \text{KoS}) - \text{min}(\text{Schl}, \text{KoS})} \\ &= \frac{123 - 100}{173 - 100} = 0.315 = 31.5\% \end{aligned} \quad (3.4)$$

3.2.3 Aggregated Support

Equation 3.5 shows how the aggregated support for a user attribute value is calculated. The aggregated support is the support calculated for a specific user attribute value based on the supports given by users when they answered a micro-task regarding this user attribute or when they evaluated the user attribute of the item. The aggregated support is needed to sum up the suggestions by the community about this user attribute value. The aggregated support is calculated by summing up the support for the user attribute values of all users who evaluated the item and then related to the total number of evaluations of the corresponding user attribute. This is shown in Equation 3.6.

$$\text{support}_{\Sigma}(i, ua, uav) = \frac{\sum_{u \in U} s(i, u, uav)}{N_s(i, ua)} \quad (3.5)$$

As mentioned above, $N_s(i, ua)$ is the number of evaluations for a specific user attribute. The reason why this count is used to normalize the aggregated support and not just the number of evaluations for that specific user attribute value, is explained with a simple example. If, for example, for the item *Kitzbühel* for the user attribute *Price level* in *Skiing Resorts*, the user attribute value *expensive* was only evaluated once with a support of 100% and the user attribute value *moderate* was evaluated four times (all four specified a support of 80%) *expensive* would be considered as the better fitting user attribute value. But when using the total number of evaluations of a given user attribute as a normalization factor, the number of evaluations for a user attribute value influences the aggregated support for this user attribute value. For this example, *expensive* would result in an aggregated support of $100\%/5=20\%$, whereas for *moderate* the aggregated support would be $80\%*4/5 = 64\%$.

$$N_s(i, ua) = \sum_{u \in U} [\exists uav \in ua \wedge s(u, i, uav) \neq NULL] \quad (3.6)$$

The following equations will show an example of the computation of the aggregated support for the item *Schladming*(Schl) and the user attribute *Target Audi-*

ence(TA). As this user attribute has several user attribute values, the aggregated support for each user attribute value needs to be calculated. First, the number of evaluations for the user attribute *Target Audience* is counted and then the aggregated support for the user attribute value *recreational athlete*(RA):

$$N_s(Schl, TA) = \sum_{u \in U} [\exists uav \in TA \wedge s(u, Schl, uav) \neq NULL] = 2 \quad (3.7)$$

$$\text{support}_{\Sigma}(Schl, TA, RA) = \frac{\sum_{u \in U} s(Schl, u, RA)}{N_s(Schl, RA)} = \frac{80\%}{2} = 40\% \quad (3.8)$$

Now, the calculation of the aggregated support for the user attribute value *family*(FM) is done:

$$\text{support}_{\Sigma}(Sch, TA, FM) = \frac{\sum_{u \in U} s(Sch, u, FM)}{N_s(Sch, FM)} = \frac{90\% + 80\%}{2} = 85\% \quad (3.9)$$

Finally, the aggregated support for the user attribute value *snowboarder*(SB) is:

$$\text{support}_{\Sigma}(Sch, TA, SB) = \frac{\sum_{u \in U} s(Sch, u, SB)}{N_s(Sch, SB)} = \frac{80\%}{2} = 40\% \quad (3.10)$$

When looking at the resulting values, the above mentioned relation between the actual support value and the number of evaluations for a specific user attribute value can be seen. The aggregate support for *family* is much higher than for *snowboarder* even if the support values for the different evaluations is similar. As there are two evaluations for *family* and only one for *snowboarder*, *family* gets a higher aggregated support as more users evaluated it. When just the average would have been calculated, the information about the number of evaluations would have been gone.

3.2.4 Utility Function

For the ranking of the items in the result set, a utility function is used. Equation 3.11 defines the used utility function. To compute the ranking of the items, all supports for each requirement value are summed up and weighted with the factors $w(uav)$ and $w(ia)$. Those weights can either be learned by machine learning algorithms (such as genetic algorithm [37]) or specified by the user. The weighting factors for

the examples in this thesis are set to 1. The utility function values do not have a unit. After calculating the utility for all items in the result set, the items are ranked by the value of the utility function from highest to lowest. Table 3.11 gives the supports for each attribute and item and the corresponding utility values based on the given requirements. The item *Schladming* (Schl) has the highest utility value for the given requirements and is therefore ranked as first item in the result set.

$$\begin{aligned} \text{utility}(item, REQ) = & \sum_{uav \in REQ} \text{support}_{\Sigma}(i, uav) \times w(uav) \\ & + \sum_{ia \in REQ} \text{support}_{\Sigma}(i, ia) \times w(ia) \end{aligned} \quad (3.11)$$

Item	Price Level	Target Audience	Family Friendly (=Yes)	Kilometers of Slopes (=123)	Utility Value
Kitzbühel	exp.(40%) mod.(45%)	rec. athlete(70%) top. athlete(35%) snowboarder(40%)	100%	100%	285
Schladming	mod.(75%)	rec. athlete(40%) family(85%) snowboarder(40%)	100%	31.5%	331.5
Obertauern	cheap(70%)	rec. athlete(70%) family(65%)	100%	0%	165

Table 3.11: Support values for all items in the result set. Based on the requirements in Table 3.9, the utility function was calculated for all three items. For Price Level and Target Audience the aggregated support for each evaluated user attribute value is shown. Family Friendly and Kilometers of Slopes show the support for the item attributes. The value of item Obertauern for Kilometers of Slopes is lower than the specified requirement of 120 km, therefore the support is 0.

As an example, the computation of the utility function for the item *Schladming*

is shown. The weighting factors $w(uav)$ and $w(ia)$ are set to 1 for simplification.

$$\begin{aligned}
\text{utility}(\text{Schl}, \text{REQ}) &= \sum_{uav \in \text{REQ}} \text{support}_{\Sigma}(\text{Schl}, uav) \times 1 \\
&+ \sum_{ia \in \text{REQ}} \text{support}_{\Sigma}(\text{Schl}, ia) \times 1 \\
&= \text{support}_{\Sigma}(\text{Schl}, \text{moderate}) \times 1 \\
&+ \text{support}_{\Sigma}(\text{Schl}, \text{family}) \times 1 \\
&+ \text{support}_{\Sigma}(\text{Schl}, \text{snowboarder}) \times 1 \\
&+ \text{support}_{\Sigma}(\text{Schl}, \text{FamilyFriendly}, \text{Yes}) \times 1 \\
&+ \text{support}_{\Sigma}(\text{Schl}, \text{KilometersofSlopes}, 120) \times 1 \\
&= (75 + 85 + 40) \times 1 + (100 + 31.5) \times 1 \\
&= 200 \times 1 + 131.5 \times 1 = 331.5
\end{aligned} \tag{3.12}$$

Table 3.11 shows the calculated utility values for all three items. Now the items can be ranked according to the utility values. The final ranking of the items can be seen in Table 3.12 where the item fitting the requirements best is *Schladming* because it has the highest utility value.

Rank	Item
1	Schladming
2	Kitzbühel
3	Obertauern

Table 3.12: Final ranking of the items after calculating the utility values for each item.

4

PeopleViews System Description

In this chapter, the technical aspects and structures as well as the PEOPLEVIEWS user interface will be discussed in detail. First of all, the used architecture will be presented. Furthermore, the different interfaces of the system will be shown and different types of micro-tasks and their usage will be described. The game included in PEOPLEVIEWS will be explained.

4.1 PeopleViews Architecture

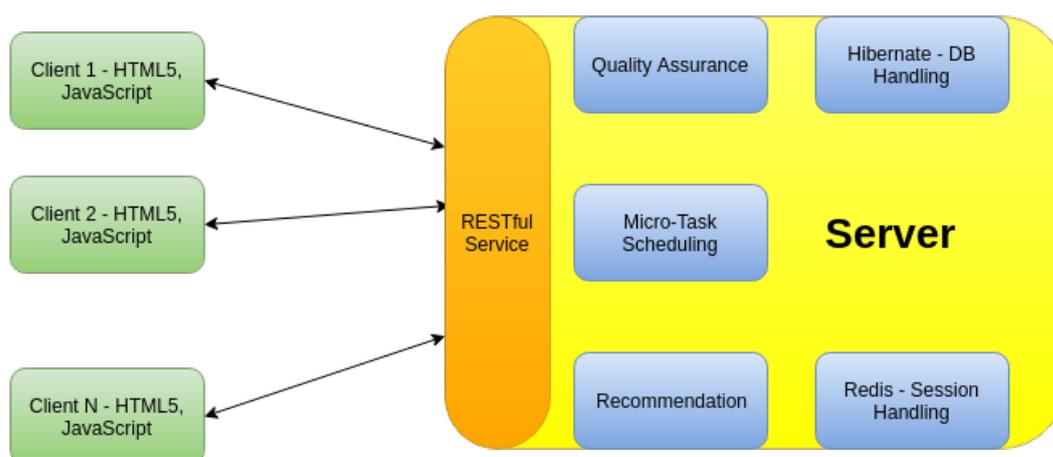


Figure 4.1: Overview of the PEOPLEVIEWS architecture. The decoupling of user interface and server can be seen.

Figure 4.1 shows an overview of the PEOPLEVIEWS system architecture. The two blocks *Quality Assurance* and *Recommendation* were implemented in two other master's thesis.

In principle, the *Quality Assurance* component analyzes the data generated using micro-tasks or evaluations. Furthermore, it keeps track of newly added items. For example, if a new item *Leogang* in the recommender *Skiing Resorts* would be added, it is necessary to collect support values regarding the user attribute values for the item to appear in recommendations (with no provided supports all aggregated support values are zero, which means that the item does not appear in recommendations). To initiate the collection of support values, the *Quality Assurance* component provides information to the micro-task scheduling algorithm regarding the items and user attribute values that need to be processed by users utilizing micro-tasks.

The *Recommendation* component calculates recommendations according to the requirements specified by the user. To rank the items accordingly, it aggregates the supports specified by users through micro-tasks, evaluations, and games. The basic functionality of the *Recommendation* component was already described in Section 3.

The PEOPLEVIEWS system is using RESTful web services [38]. Requests to the server are done via HTTPS but instead of calling functions on the server, REST uses resources. Resources have to be unique for each website or service that needs to be called and are identified via the unique URL of the resource. The resource can have any file format such as XML, PNG, or MP3. To perform operations on one of the resources, the URL and the HTTP method need to be sent to the server. For example, one can send a GET request for the URL `https://peopleviews.com/addItem`, which would display the interface for adding a new item in the user's browser. One main advantage of RESTful services is that it can run on any server. This guarantees a high scalability of RESTful services.

The server was implemented using Java and the additional library *Spark* [39] for Machine Learning and Genetic Algorithms to implement the recommendation algorithms as those rely on such techniques.

On the server, a database is used to store the data sent by the client. The application uses Hibernate [40] which uses Java classes and maps them directly to tables in the database. Therefore, no additional mapping or wrapper for the classes to store them in a database are needed.

To build the web application itself, the Spring Framework [41] was used. This framework is perfectly suitable to create RESTful web services using the MVC architecture (Model-View-Control).

To perform session handling, Redis [42] was used. Redis is a server which can be used as a database as well, but as it just holds the whole data in the memory

it is not suitable to be used as a data storage for a longer time. Session handling, however, is a perfect application for this system.

The user interface was implemented in HTML5 and JavaScript. The client sends every request using asynchronous HTTPS requests via JavaScript. The previously described RESTful resources are used in this case. Bootstrap [43] was used as a framework for the user interface creation. It contains many components such as tables or buttons which can be used when creating user interfaces and was initially developed by Twitter. The JavaScript library jQuery [44] was used in addition to bootstrap. The library offers components which support the developer in implementing features such as asynchronous requests, data handling, or binding data objects to UI components.

4.2 User Interfaces

In Figure 4.2, the home screen after logging in can be seen. The menu on the left contains all necessary options to interact with the recommender. In the middle of the screen, the recently most popular recommenders are shown.

The lower part of Figure 4.2 shows a table of the recommenders the user is most active in. Activity in a recommender includes, for example, adding or evaluating items. The table also contains the *PV Points* of each recommender. These are the PEOPLEVIEWS POINTS the logged in user already achieved in the recommender. PEOPLEVIEWS POINTS are used to motivate people to interact with the system. Table 4.1 shows how many points a user gets for which action:

Action	PV Points
Create Recommender	50
Create Item	10
Evaluate Item	2
Solve Micro-Task	1
Play Game	2

Table 4.1: PEOPLEVIEWS POINTS a user gets for a specific action.

On the lower right side of Figure 4.2, the current rank and the history of the users rank depending on the sum over all PEOPLEVIEWS POINTS of all recommenders can be seen. This graph is also used to motivate users to interact more with the system to gain more PEOPLEVIEWS POINTS and achieve a higher rank.

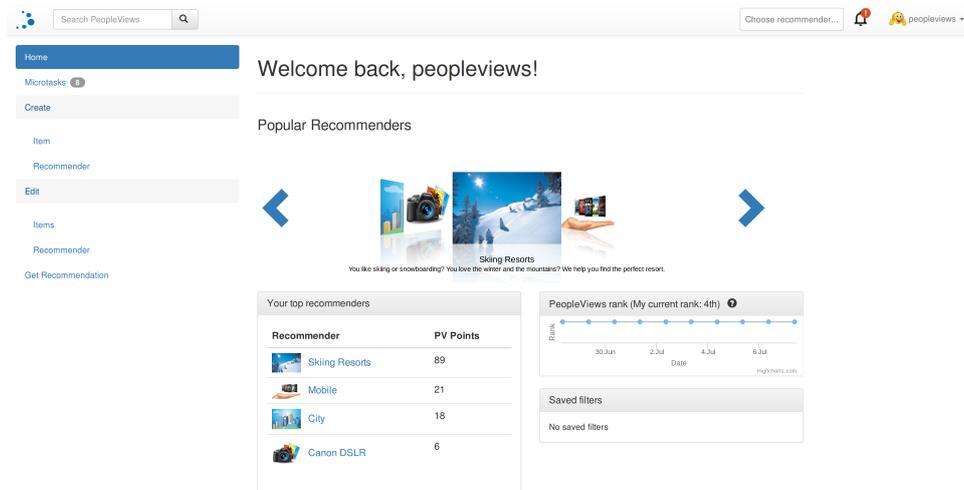


Figure 4.2: The home screen after logging in to PEOPLEVIEWS. The lower left corner contains a list of recommenders the user is most active in. The lower right corner shows the current rank and rank history of the user. The list on the left side shows the navigation of PEOPLEVIEWS.

4.3 Recommender Interfaces

As the interfaces for adding and editing a recommender are almost equal, only one of the two will be described here. Due to the size of the interface it is split into three parts to make it easier to describe. Figure 4.3 shows the first part of the interface when adding a recommender to the system.

Examples of the interface when adding item attributes can be seen in Figure 4.4. The item attributes shown in that screenshot were taken from the example recommender defined in Table 3.3. The first attribute (*Cross-Country Skiing*) was defined as having the input type *Enumeration* and two different choices (*Yes*, *No*). Furthermore, the similarity measure *Exact Match* was chosen here. The definitions of all similarity measures was given in Section 3.2.2. Compared to the second item attribute, an additional input field is displayed to enter the choices for the enumeration.

The second item attribute (*Kilometers of Slopes*) shows the interface for an attribute with input type *Number* and a similarity measure of *More is better (MIB)*.

Add Recommender

Recommender name

Image



Description

Tags (comma separated)

Figure 4.3: User interface for adding basic recommender information. The same screen is used for editing a recommender.

Cross-Country Skiing

Enumeration ▾

Exact match ▾

Is cross-country skiing possible there?

Usable as recommendation filter
?

Possible answers (comma separated)

Yes x

No x

Kilometers of Slopes

Number ▾

More is better ▾

How many kilometers of slopes does the skiing resort have?

Usable as recommendation filter
?

Figure 4.4: The interface for adding item attributes to the recommender. Item attribute name, type, similarity measure and question asked to the user who creates an item need to be specified.

In Figure 4.5, the third and last part of the interface for creating or editing a recommender can be seen. This part shows the interface for adding a user attribute to the system. The *attribute name* is used in the recommendation screen later to restrict the list of recommendations. The question is displayed to users when they evaluate an item from the recommender. With the button *Add answer* on the bottom of the interface a user can add as many user attribute values as she needs. The check box (*Allow multiple answers*) in the lower right area has an impact on the evaluation. If this check box is checked, evaluating users are allowed to enter supports for more than one user attribute value of the user attribute. When the box is not checked, users can only evaluate one user attribute value for this question. This option also influences the micro-tasks. If a user attribute is marked to allow multiple answers the generated micro-task is different to one generated for a single answer user attribute. The different micro-tasks created depending on this check box will be described in detail in Section 4.5.

User attributes... ?

▼ Price Level

Attribute name

Price Level Remove

Question to user when evaluating item

How would you evaluate the price level in this skiing resort?

Possible answers

expensive Remove

moderate Remove

cheap Remove

+ Add answer Allow multiple answers

Figure 4.5: Adding a user attribute to a recommender. Attribute name, question asked to users which evaluate the attribute and possible answers need to be specified here. Furthermore, users need to define if multiple answers for this user attribute are allowed.

The interface in Figure 4.6 shows the screen where the creator (owner) of a recommender can add experts to the recommender. Table 4.2 lists the actions each type of user is allowed to take in PEOPLEVIEWS.

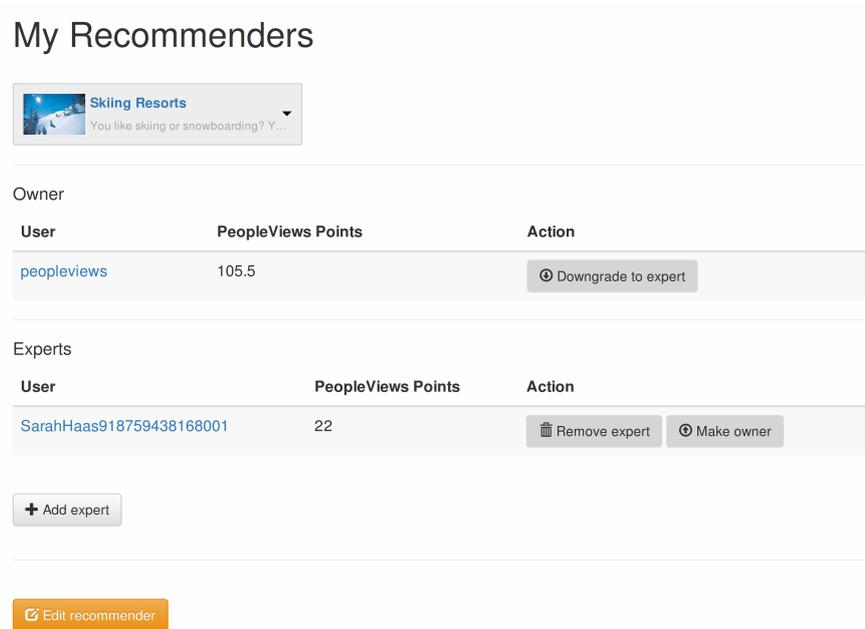


Figure 4.6: Interface for adding or deleting experts. The owner (creator) of the recommender can also give owner rights to an expert or remove experts.

User Type	Allowed Actions
Anonymous user	Get recommendations, view items, compare items
Logged-in User	Same as anonymous user plus adding items, editing items the user created, evaluating all existing items, answering micro-tasks, adding recommenders (user becomes owner of that recommender)
Expert	Same as logged-in user plus edit all recommenders where the user is expert in
Owner	Same as logged-in user plus adding experts to own recommenders, upgrading existing experts to owners, downgrading existing owners to experts, removing experts, editing own recommenders

Table 4.2: List of actions users can take depending on their user type.

4.4 Item Interfaces

As the interfaces for adding and editing an item are almost identical, only one of them will be shown here. Figure 4.8 shows the interface when editing an item. The item attributes need to be entered by the user who creates or edits the item (privileges of different types of user can be seen in Table 4.2). Furthermore, the description, tags, picture and photo need to be added. In the lower part of the

screen a check box is shown. This check box can be unchecked to prevent the item from being published. This can be used if, for example, the user does not have all needed information yet to create the item. If the user unchecked this check box, the item cannot be seen by other users. Later when the user is able to add the previously missing information, she can edit the item and check the check box to make the item visible to all other users.

The interface in Figure 4.7 shows the evaluation screen for an item. Users can specify supports for every user attribute and corresponding user attribute values defined in the corresponding recommender. When evaluating the item name, image and description are shown.

Evaluate »Kitzbühel«

Recommender "Skiing Resorts"



Description

Kitzbühel is one of the most legendary winter sports towns in the Alps and is measured only at the top. Around Hahnenkamm, Streif and Mausefalle are guests 60 challenging pistes with a total of 173 kilometers to feet. Perfect for children and families. A very large continuous skiing area.

▶ How would you evaluate the price level in this skiing resort?

▶ Which experience level should you have to go skiing there?

▶ How would you evaluate the quality of lift facilities?

▼ What target audience is the skiing resort constructed for?

family		? 0%	100%
recreational athlete		? 0%	100%
top-class athlete		? 0%	100%
snowboarders		? 0%	100%

▶ How would you evaluate the quality of skiing lodges?

Cancel

Save

Figure 4.7: Interface when evaluating an item. Item name, image and description are shown. Supports for the user attribute values of each user attribute can be specified.

Edit item

Item name

Image



Description

Kitzbühel is one of the most legendary winter sports towns in the Alps and is measured only at the top. Around Hahnenkamm, Streif and Mausefalle are guests 60 challenging pistes with a total of 173 kilometers to feet. Perfect for children and families. A very large continuous skiing area.

Tags (comma separated)

Link

Item attributes

Is this skiing resort family friendly?

Can you do sledging there?

Is cross-country skiing possible there?

How many kilometers of slopes does the skiing resort have?

Which types of lift facilities are available?

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

Figure 4.8: Interface for adding or editing an item. Textual information such as item name, description, tags and link need to be added. Furthermore, the item attributes need to be specified.

4.5 Micro-Task Interfaces

There are six types of micro-tasks currently implemented in PEOPLEVIEWS. To distinguish between them they are referenced to as *micro-task type X* or *micro-task of type X* or *type X micro-tasks*, where X is the number. Each type is designed to gather a different kind of information depending, for example, on the previously mentioned check box that allows multiple answers. In this section, the different types will be explained in detail. The selection criteria for each type will be discussed in Section 5.3.

Micro-tasks of type 1 as shown in Figure 4.9 evaluate the support for exactly one user attribute value of a specific user attribute of an item. These micro-tasks are used, for example, if the users, who already evaluated the user attribute value did not agree on a similar support value for the user attribute value. For example, two users entered a high support around 90% and two other users entered a low support around 10%. To be sure which support is correct, more data is needed and therefore, this kind of micro-task is distributed. The detection of such items and attributes is done by the quality assurance component of PEOPLEVIEWS which is discussed in a master's thesis by Michael Schwarz.

The screenshot shows a web interface for a micro-task. At the top, it says 'Skiing Resorts'. Below that is a small image of a ski lodge. To the right of the image, the text reads: 'Item »Kitzbühel«, attribute »Quality of skiing lodges«: How well does it support »good«?'. Below this is a horizontal slider. The left end is labeled 'good' and has a blue dot with a question mark below it. The slider is marked with '0%' and '100%'. At the bottom left, there is a checkbox labeled 'Don't show questions for this recommender'. At the bottom right, there are two buttons: 'Skip' with a red 'x' icon and 'Next' with a blue checkmark icon.

Figure 4.9: Example for a type 1 micro-task. The user needs to specify the support for the user attribute value of one user attribute for a specific item.

In Figure 4.10, a micro-task of type 2 can be seen. The user needs to choose the better suiting item and enter the support for the user attribute value of the chosen item. This type is similar to type 1 but in this case two items are shown to the user. Both items and the user attribute are selected based on the same criteria as for micro-tasks of type 1. By allowing the user to select one of two items instead of showing one fixed item the possibility of a user having knowledge about one item is doubled.

The screenshot shows a window titled "Skiing Resorts" with the question: "Which item fits the answer »moderate« of the attribute »Price Level« better?". Below the question are two image cards: "Kitzbühel" (left) and "Obertauern" (right). Below the cards is a slider control with a blue dot at the 0% mark, labeled "moderate" on the left and "0%" and "100%" on the right. At the bottom left is a checkbox labeled "Don't show questions for this recommender". At the bottom right are two buttons: "Skip" (with an 'x' icon) and "Next" (with a checkmark icon).

Figure 4.10: Example micro-task of type 2. The user has to choose one item and estimate the support for the given user attribute value.

Figure 4.11 represents a micro task of type 3. This micro task is generated for all user attributes where the *Allow multiple answers* check box was not checked when creating the user attribute. The user needs to choose one user attribute value and then enter the support for the chosen user attribute value.

Figure 4.12 shows a micro-task of type 4 which is needed if a user attribute is allowed to have multiple answers. This micro-task is used to get support values for one or more user attribute values of a specific user attribute for a specific item. This is used when there are very few evaluations for this user attribute-item combination in the system and the confidence in the support would be very low.

Skiing Resorts



Item »**Kitzbühel**«: Which answer fits the attribute »**Quality of lift facilities**« best?

new
 average
 old

How well?

Don't show questions for this recommender

Figure 4.11: Example of a micro task of type 3. The user has to choose one single user attribute value and specify the support for the chosen one.

Skiing Resorts



How would you evaluate the attribute »**Target audience**« for the item »**Saalbach Hinterglemm**«?

family
 recreational athlete
 top-class athlete
 snowboarders

Don't show questions for this recommender

Figure 4.12: Example micro-task of type 4. The user can estimate the support for one or more user attribute values.

Figure 4.13 displays a micro-task of type 5 which is actually a form of CAPTCHA. Those micro-tasks are used to determine whether a user is human or not. As people implement malicious software, named bots, that automatically interact with web sites it is important to be able to distinguish between human and non-human users. As data created by bots is not reliable, it is important to be sure that the user who, for example, creates a new evaluation is actually a human. CAPTCHAs of this kind are very hard for computers to solve but easy for humans as the recognition of items in a picture is very hard for computers but very easy for humans. To express the percentage of users being human, a *human score* is introduced by the quality assurance component. Each user gets human score points when interacting with PEOPLEVIEWS. Interactions such as correctly answering CAPTCHAs are awarded with human score points. The sum of human score points is weighted with the human score of the users having the highest human scores so that the human score ranges between 0 and 1. The computation of the human score is given in equation 4.1.

$$HS = \min \left(1, \left(\sum_{p \in \text{points}} p \right) \cdot \frac{\sum_{u \in \text{topuser}} HS(u)}{|\text{topuser}|} \right) \quad (4.1)$$

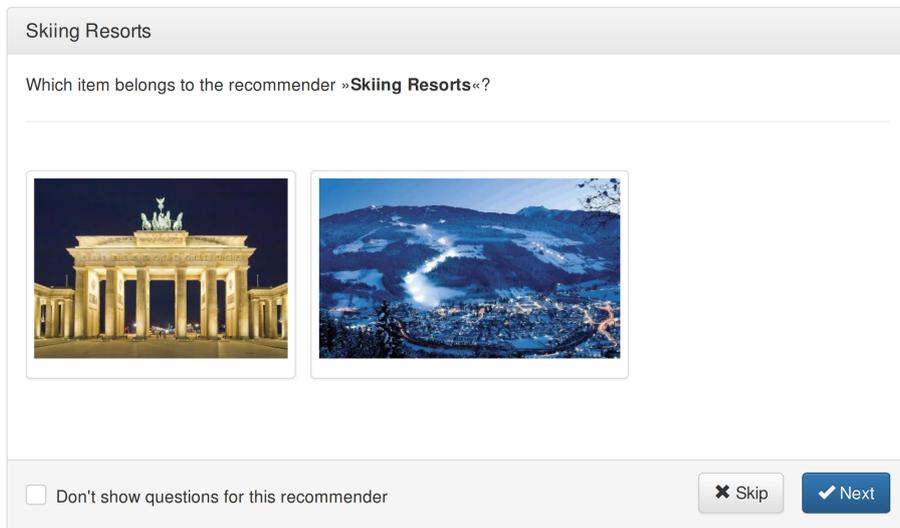
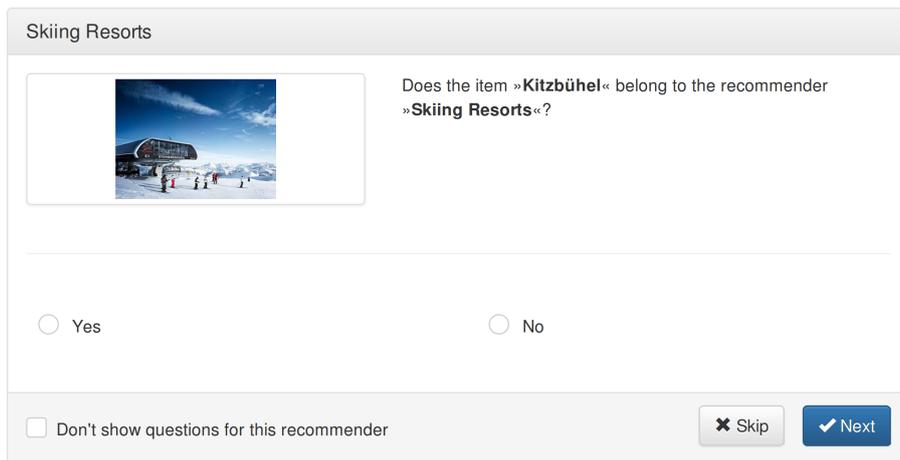


Figure 4.13: Example of a micro-task of type 5. The user has to choose the image related to the given recommender.

As every user can add a new item to the system, it is necessary to check if the item was created in the correct recommender. An example of a item in a wrong recommender would be to add the skiing resort *Kitzbühel* to the recommender *Mobile Phones*. This is done with a micro task of type 6 which can be seen in

Figure 4.14. Those micro-tasks are always distributed when a new item is added to a recommender to check if it really belongs there. If more than 50% of people think the item was added to the wrong recommender the item is removed as it apparently should have been added to another recommender.



The screenshot shows a user interface for a micro-task. At the top, the title 'Skiing Resorts' is displayed. Below the title, there is a placeholder for an image, which contains a photograph of a ski resort building. To the right of the image, the text asks: 'Does the item »Kitzbühel« belong to the recommender »Skiing Resorts«?'. Below the question, there are two radio buttons: 'Yes' and 'No'. At the bottom of the interface, there is a checkbox labeled 'Don't show questions for this recommender', a 'Skip' button with a close icon, and a 'Next' button with a checkmark icon.

Figure 4.14: Example of a micro-task of type 6. The user has to choose whether the given item is related with the given recommender.

4.6 Recommendation Interfaces

Recommendations generated by the *Skiing Resorts* recommender can be seen in Figure 4.15. On the left the filter criteria can be set. Setting criteria changes the number of items in the list and also their ordering according to the chosen constraints.

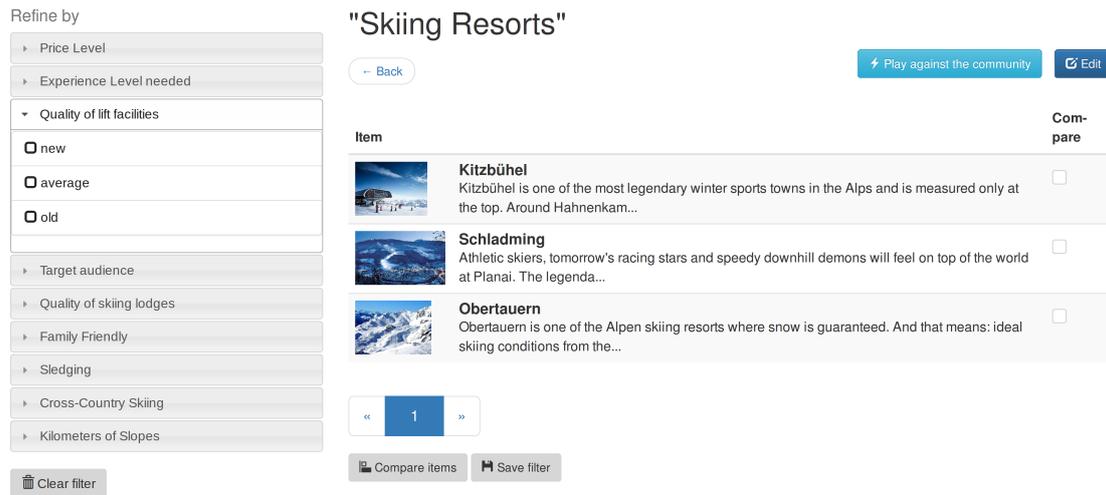


Figure 4.15: Example of the recommendation screen for a recommender. The user can choose an item and view the details or compare two items.

Users can also compare two items to each other. This can be done by checking the check boxes of 2 different items on the right of the recommendation screen and then pressing *Compare items* on the bottom. The resulting interface would look like the one in Figure 4.16. In the middle of the screen, a spider web representation of the supports for the user attribute values of the different user attributes can be seen³. On the bottom, the item attributes are shown. The item attribute *Kilometers of Slopes* of the item *Kitzbühel* is marked because the order relation of this item attribute is *More is better (MIB)*. As the value for *Kitzbühel* is higher than for *Schladming*, it is marked in this example. The other item attributes do not have order relations due to their input type and therefore are not marked.

³ In this example, only two of the spider webs are shown as the others look similar.

Comparison

[← Back](#)



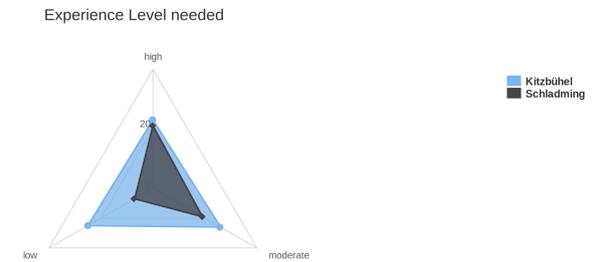
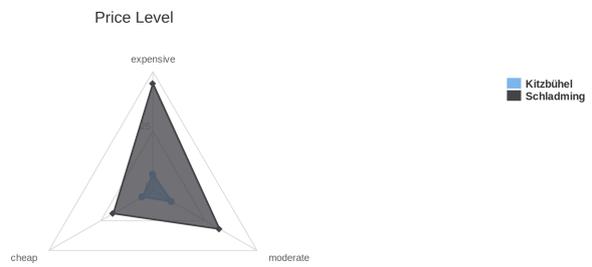
Kitzbühel

24



Schladming

76



Cross-Country Skiing	Yes
Kilometers of Slopes	173
Types of Lift Facilities	cable way, chair lift, drag lift

Cross-Country Skiing	Yes
Kilometers of Slopes	123
Types of Lift Facilities	cableway, chair lift, drag lift

Figure 4.16: Example of the comparison screen for two items. The spider webs give information about the user attributes. The list at the bottom of the screen are the item attributes.

4.7 Game

The system contains a game with a purpose which is used to acquire data and motivate users to interact with the system. The user selects a recommender and can then play the game containing questions regarding the selected recommender. This means that the user will only get questions concerning the chosen recommender. In Figure 4.15 the *Play against the community* button can be seen in the upper right corner.

The game consists of 10 questions the user needs to answer. The questions ask users to specify the support of a specific user attribute and user attribute value for a given item. Game questions never contain multiple possible answers where the user has to choose one or more. Figure 4.17 shows an example of such a question. The user needs to guess the support given by the community for a specific user attribute value of an item. For each question, the user can get 10 points at most. The better the user's answer matches the support given by the community, the more points the user gets for this question. In Table 4.3, the scaling of the point reduction, depending on the absolute difference between the users estimated support and the support given by the community, is shown.

Absolute Difference	Reduction in %
≤ 5	0
6 - 10	5
11 - 20	15
21 - 40	25
41 - 60	50
61 - 80	75
81 - 100	100

Table 4.3: Reduction of points per question depending on how large the absolute difference between the users answer and the answer of the community is.

The computation of the difference is rather easy. To compute it, the support entered by the user $support_U(i, ua, uav)$ and the aggregated support $support_\Sigma(i, ua, uav)$ for the item-user attribute-user attribute value combination are needed. Equation 4.2 shows the computation of the absolute difference d :

$$d = |support_\Sigma(i, ua, uav) - support_U(i, ua, uav)| \quad (4.2)$$

This value d can then be used to calculate the point reduction for each question as it is shown in Table 4.3.

As users might cheat in the game and open another instance of the system to have a look at the support given by the community, the game contains a time constraint to overcome this problem. The time a user may need to answer one question is set

Guess what the community thinks!

Recommender "Skiing Resorts"

Questions answered 1/10

How well does the following question-answer combination match for »**Schladming**«?

How would you evaluate the price level in this skiing resort?

cheap

0% 100%

[→ Next](#)

Figure 4.17: An example of the interface of a game question. The user needs to specify the support for the given user attribute value.

to 15 seconds as this is enough time to read the question and quickly think about the correct answer. If a user is not able to solve each question within 15 seconds, the points she achieved by answering the questions will be reduced. Table 4.4 shows the scaling of this reduction depending on how much more time a user needed to finish. The total points are then calculated from the points acquired by the answering of questions and the optional reduction because of exceeding the time constraint.

Time extension in s	Reduction in %
≤ 10	5
11 - 25	10
26 - 35	25
36 - 45	50
46 - 60	75
≥ 61	100

Table 4.4: Reduction of points in percent is shown depending on how much more time than the estimated 150 seconds (10 questions, 15 seconds each) the user needed.

After finishing the game, the user receives a game summary that compares the support estimated by her and the support given by the community for each question as well as the points she got for each question and the eventual reduction because of exceeding the time. An example, thereof can be seen in Figure 4.18. The points a

user achieved are then added to the users so called game points she already received when playing games in this recommender before. The user will see a high score and his current rank compared to all other users who played games in this recommender.

Summary



Item	Attribute name	Attribute value	Your answer (👤) vs. Correct answer (🎯)	Game score
Schladming	Price Level	cheap		0
Schladming	Quality of lift facilities	average		0
Kitzbühel	Quality of skiing lodges	good		0
Schladming	Target audience	top-class athlete		0
Schladming	Quality of skiing lodges	good		0
Schladming	Quality of skiing lodges	moderate		0
Kitzbühel	Experience Level needed	moderate		0
Schladming	Target audience	family		0
Kitzbühel	Quality of lift facilities	old		1
Schladming	Price Level	moderate		0
Points reduced by 50%				-0.25
Total game score				0.5

Show HighScore List

Figure 4.18: This image shows the interface of the game summary. The reduction of points due to exceeding the time limitation is displayed before the total game score.

Before the user can play the game, the system checks if there is enough data available to play the game. Data, in this case, means that there must exist at least 10 different user attribute values that were already evaluated in this recommender to compare the answers given by the user with the answers given by the community. If there are less than 10 different ones, the game cannot be played as a user would then get at least two equal questions which does not make sense. Another restriction for the selection of questions is, that users will not get questions for user attribute value-item combinations for which they already specified a support.

For clarification, the process of playing the game is again illustrated here:

1. Check if the recommender contains support values for at least 10 different user attribute values.

2. Select 10 random user attribute values and create questions from them.
3. Let the user guess the support in each given question one after the other.
4. Compute the difference between the support estimated by the user and the support given by the community.
5. Calculate the total points depending on the differences.
6. Check how long the user needed to solve each question and reduce points if necessary.
7. Calculate the points and show the summary.
8. Add the points to the existing game points and show the high score and the users rank.

5

Micro-Task Scheduling Approach

In this chapter, the micro-task scheduling algorithm will be described in detail. As already discussed in Sections 2.3 and 3.1, micro-tasks consist of short questions regarding a specific item and user attribute and are used to collect support values regarding user attribute values. The micro-task scheduler's purpose is to find users who might be best suited to answer those questions. It is assumed that better suiting users are more likely to answer micro-tasks.

One aspect of the micro-task scheduling approach is to match users with items based on keywords extracted from item information such as item description or tags assigned to the item. A history of user interactions such as evaluating or viewing items is kept in a user profile. For all items the user previously interacted with, keywords are extracted. As already discussed by Pazzani et al. [9], extracting keywords from unstructured text such as item descriptions can be used to create user profiles. Those profiles can later be used to find relevant items for the user. The applied extraction and weighting algorithms will be explained in this chapter.

Furthermore, micro-task scheduling needs to be invoked by another component. The *Quality Assurance* component (see Section 4.1) analyzes existing data and invokes *Micro-task Scheduling*. It provides information on which item and user attribute values need to be considered.

To generate and distribute micro-tasks, several metrics such as a users workload or the importance of a micro-task are applied. All influencing factors will be discussed in detail.

5.1 TF-IDF

As already stated, keywords can be extracted from unstructured text such as item descriptions. After extracting keywords, weights are assigned to each keywords indicating the importance of the respective word compared to all other extracted keywords. Those importance factors are later used when matching user profiles to items which will be further explained in Section 5.4.

TF-IDF [10] means *Text Frequency Inverse Document Frequency* and can be used to assign weighting factors to words. The TF-IDF is used to calculate how important a word is to a document in a collection of documents. The calculated value depends on how often a word appears in one document and also in how many documents it appears in the corpus. This technique will be used later to assign weightings to item keywords and user keywords in PEOPLEVIEWS recommenders.

The TF-IDF value is higher if a term appears very often in a small subset of documents. A high TF-IDF value means that the term is generally important to the whole document set. Therefore, a small value means that the term is less important. Importance, in the context of TF-IDF, is measured by the IDF metric which will be explained in Section 5.1.2. This means that the words are weighted according to their importance in the whole set of documents not only by their importance within one document. Specific examples for the TF-IDF calculations will be shown later where the weighting of the item and user keywords is described in detail.

The TF-IDF of a term t_i is generally calculated by multiplying the TF and the IDF value of the given term t_i with each other. This can be seen in Equation 5.1.

$$TF \times IDF(t_i) = TF(t_{i,d}) \times IDF(t_i) \quad (5.1)$$

In the following sections, the weighting scheme used in PEOPLEVIEWS will be described. The given formulas for TF and IDF in the following two sections where implemented without using any additional library.

5.1.1 TF

TF represents the term frequency of a specific word within a document, i.e. how often the term appears in the given document. In Equation 5.2, the general formula for the calculation of the term frequency of a term t_i in a document d is given, where $N_{i,d}$ is the number of occurrences of term t_i in document d .

$$TF(t_{i,d}) = N_{i,d} \quad (5.2)$$

5.1.2 IDF

The inverse document frequency represents the importance of a term. To calculate the IDF, the total number of documents is divided by the number of documents that contain the term t_i . Additionally, the logarithm is applied to the quotient which gives the logarithmically scaled inverse document frequency of the term t_i . Equation 5.3 illustrates the general formula to calculate the IDF, where N is the total number of documents and N_i is the number of documents containing term t_i .

$$IDF(t_i) = \log \left(\frac{N}{N_i} \right) \quad (5.3)$$

5.2 Extraction and Weighting of Keywords

In this section, the extraction and weighting of user and item keywords will be described. The calculation of micro-task and recommender keywords will also be shown. First, the extraction of keywords will be shown for both items and users. Afterwards, the weighting of the keywords for both will be described in detail. Finally, the generation of micro-task and recommender keywords will be described. For the extraction as well as the weighting, examples will be shown to clarify the theoretically described concepts.

Tables 5.1, 5.2, 5.3 and 5.4 contain the data sources for the extraction of recommender and item keywords as well as user keywords and micro-task keywords.

Data Sources for Recommender Keywords
Recommender description of the recommender
Recommender tags of the recommender

Table 5.1: Data sources used to extract keywords of a recommender.

Data Sources for Item Keywords
Item description of the item
Item tags of the item

Table 5.2: Data sources used to extract keywords of an item.

5.2.1 Keyword Extraction for Items

The keywords for the items are extracted from the item description and the tags assigned to the items. Whenever an item is created or changed, the old keywords

Data Sources for User Keywords
Recommender keywords (see Table 5.1) of each recommender the user interacted with
Item keywords (see Table 5.2) of each item the user interacted with

Table 5.3: Data sources used to extract keywords of the user profile.

Data Sources for Micro-Task Keywords
Recommender keywords (see Table 5.1) of the corresponding recommender in the micro-task
Item keywords (see Table 5.2) of the corresponding item in the micro-task

Table 5.4: Data sources used to extract keywords of a micro-task.

are deleted and newly generated to keep the item keywords up to date.

To extract keywords, the description text and tags are separated into single words to be able to process each word and decide whether it is suitable as a keyword or not. After separating the keywords, a stop-word removal is performed. Stop-words are, for example, articles, conjunctions, adverbs, and many more. Stop-words contain no necessary information when it comes to micro-task scheduling. Therefore, they can be removed from the list of possible words. All possible punctuation marks are also removed as they do not contain important information. Words containing punctuation marks such as dashes are split into two words. Moreover, numbers are removed because they are also not useful in the scheduling process. After removing every unnecessary and clueless information, the words are uncapitalized to be able to match words with lower and upper case letters. The set of words left, after performing all previous steps, are the final keywords for the given item.

Stemming will not be performed on the text. Stemming means that words are transformed to their root form, for example, words in plural are transformed to singular. Peng et al. [45] state that stemming requires a significant amount of additional computation time and furthermore, decreases the precision when matching keywords.

One problem of traditional stemming is its blind transformation of all query terms, that is, it always performs the same transformation for the same query word without considering the context of the word.

The authors list an example of a transformation leading to a totally different meaning. In their example, *book store*, *book stores*, *book storing*, and *booking store* are all reduced to *book store*, although only *book stores* is a meaningful match.

In many cases, reducing plural to singular is not desired as, for example, a user might be interested in mobile phones having multiple *CPU cores* but not in mobile phones with just one *CPU core*. On the other hand, for example, the verbs *ski* and *skiing* would be reduced to *ski* which would make sense and would also be helpful in matching user profiles and item description. However, due to the initially named disadvantage, stemming will not be applied on the keywords.

The following example will show the extraction of keywords from an item description. The item description and tags of the item *Sony Xperia M5* will be used as an example text here. The tags are: *Display, Performance, Storage, Camera*. The item description is:

Sometimes you only get one chance to get the perfect shot with the camera. That's why the Xperia M5 is packed full of technology to help you capture the moment as it happens. Shoot like a pro with a 0.25-second Hybrid Autofocus, 21.5 megapixels and super-sharp zoom camera.

First of all, the item tags are added to the list of words, then the stop-word removal is performed and afterwards the punctuation marks are removed, which leads to the following words left:

get chance get perfect shot camera Xperia M5 packed full technology help capture moment Shoot pro 0 25 second Hybrid Autofocus 21 5 megapixels super sharp zoom camera Display Performance Storage Camera

Finally, the numbers are removed, the words are uncapitalized and reveal the final set of words that will be used as keywords for the item *Sony Xperia M5*:

get chance get perfect shot camera xperia packed full technology help capture moment shoot pro hybrid autofocus megapixels super sharp zoom camera display performance storage camera

5.2.2 Keyword Extraction for Users

The keyword extraction for users is similar to the one for items. The main difference here is that the user keywords are generated by using the keywords from items and recommenders the user interacted with. The interactions of users with the system are stored which enables the system to extract the item keywords and recommender keywords from items and recommenders the user interacted with. These extracted keywords can then be added to the user keywords. The keywords for recommenders are extracted using the same approach as for item keywords. The recommender description is used as a text for the extraction. With the extracted recommender

keywords and item keywords from each item, the user keywords are constructed. The user keywords simply consist of all recommender and item keywords from recommenders and items he interacted with.

In the following example, the construction of user keywords for user u_1 will be shown. The users interactions with the system can be seen in Table 5.5. The table also includes the item respectively recommender keywords. If items are marked with "- ", the recommender was just used by the user and no item was examined. In this case, only the recommender keywords can be added to the user keywords. Adding all keywords from every user interaction to a list, results in the user keywords. These user keywords can then be used, for example, to be matched with item keywords from newly added items.

Document	Recommender	Item	Recommender/Item Keywords
1	Skiing Resorts	-	like skiing snowboarding love mountains help find perfect resort winter alps rest europe
2	Skiing Resorts	Kitzbühel	most legendary winter sports towns especially skiing alps hahnenkamm streif mausefalle perfect children families continuous skiing area mountains sledging funpark top challenging pistes large measured snowboarding
3	Mobile Phones	Sony Xperia M5	get chance get perfect shot camera packed full technology help capture moment shoot pro hybrid autofocus mega pixels super sharp zoom camera display xperia performance storage camera

Table 5.5: Example interactions user u_1 made within the system. It also includes the corresponding recommender or item keywords of each interaction.

5.2.3 Weighting of Item Keywords

To generate weights for the item keywords, the previously described TF can be used. It calculates the frequency of a term in the item keywords of the given item.

An example of the TF calculation is shown in Table 5.6. For the purpose of this example the table only contains the TF calculation for some words from the given

text. The example text is:

get chance get perfect shot camera xperia packed full technology help capture moment shoot pro hybrid autofocus megapixels super sharp zoom camera display performance storage camera

Term	TF
chance	1
perfect	1
shot	1
xperia	1
hybrid	1
sharp	1
camera	3

Table 5.6: Examples of weighted item keywords using TF.

Equation 5.4 shows how to calculate the TF for the term *camera*. As the term *camera* occurs 3 times within the whole text, the result of the TF calculation is simply 3.

$$TF(t_{camera,d}) = N_{camera,d} = 3 \quad (5.4)$$

5.2.4 Weighting of User Keywords

User keywords are weighted using both TF and IDF. As already shown in Section 5.2.2, the interactions with the system lead to a set of item and recommender keywords which can be treated as user keywords. To weight these keywords, the TF-IDF is a suitable approach because each interaction leads to a document. The document is represented by the item keywords which means that each document contains the keywords of one item. From this set of documents the TF-IDF for each term can be calculated. It is possible that several items contain some equal keywords. As for every word in every document the TF-IDF is calculated, it is possible that for one word more than one TF-IDF value exists. For the weighting, however, only one weighting factor per word should be available because the matching algorithm can not deal with more than one weighting factor per word. To solve this problem, for each word that has more than one weighing factor, the weights are simply averaged. Later, examples of the TF-IDF calculation will be shown and also the averaging [46] can be seen.

In the following example, the weighting of the user keywords in Table 5.5 resulting from the TF-IDF calculation is shown.

For reasons of space limitations, Table 5.7 shows only a subset of the whole list of keywords.

Term	TF-IDF
alps	0.1761
skiing	0.2641
mountains	0.1761
children	0.4771
get	0.9542
shoot	0.4771
help	0.1761
camera	0.5283

Table 5.7: Example of weighted user keywords using TF-IDF.

The following equations give an example of the TF-IDF calculation of the word *skiing*.

Equation 5.5 shows the TF values for the term *skiing* in the different documents.

$$\begin{aligned}
 TF(t_{skiing,1}) &= N_{skiing,1} = 1 \\
 TF(t_{skiing,2}) &= N_{skiing,2} = 2 \\
 TF(t_{skiing,3}) &= N_{skiing,3} = 0
 \end{aligned} \tag{5.5}$$

Equation 5.6 gives the calculation of the IDF for the term *skiing*. The total number of documents is 3 here.

$$IDF(skiing) = \log \left(\frac{N}{N_{skiing}} \right) = \log \left(\frac{3}{2} \right) = 0.1761 \tag{5.6}$$

Now, the TF-IDF values for the different documents of term *skiing* can be calculated in Equation 5.7. As the TF for *skiing* for document 3 is zero, there is no need to calculate the IDF, as the TF-IDF value will also be zero because of the multiplication with zero.

$$\begin{aligned}
 TF \times IDF(t_{skiing,1}) &= TF(t_{skiing,1}) \times IDF(t_{skiing}) = 1 \times 0.1761 = 0.1761 \\
 TF \times IDF(t_{skiing,2}) &= TF(t_{skiing,2}) \times IDF(t_{skiing}) = 2 \times 0.1761 = 0.3522
 \end{aligned} \tag{5.7}$$

The final TF-IDF for the term *skiing* can be calculated by averaging the results for the TF-IDF values of the different documents as already described above. This

can be seen in Equation 5.8.

$$TFxIDF(t_{skiing}) = \frac{1}{2} \sum_{k=1}^2 TFxIDF(t_{skiing}, k) = \frac{0.1761 + 0.3522}{2} = 0.2641 \quad (5.8)$$

5.2.5 Extraction and Weighting of Recommender Keywords

Recommender keywords are extracted and weighted identically to item keywords. Table 5.1 lists the data sources used to extract recommender keywords. As can be seen in Table 5.3 and 5.4, recommender keywords are used in user and micro-task keywords.

5.2.6 Extraction and Weighting of Micro-Task Keywords

Table 5.4 lists the data sources used to extract keywords of micro-tasks. As can be seen there, micro-task keywords are comprised of the keywords of the corresponding item and recommender given in the micro-task.

5.3 Agenda for Micro-Task Generation

To generate a micro-task it is essential to know which recommender, item, user attribute or user attribute value the micro-task is about. This information is extracted from a so called *agenda*. The agenda is generated by the Quality Assurance component of the system which was already discussed in Section 4.1. The Quality Assurance invokes the micro-task generation and distribution by providing an agenda. An overview of that process is shown in Figure 5.1.

The agenda contains all necessary data to generate a micro-task. Each agenda has a priority between 1 and 5, where 1 is low and 5 is high, to be able to rank the agendas by their importance. An agenda with high priority is processed earlier than one with a low priority. Furthermore, an agenda has a start and an end date within which the micro-tasks for this agenda need to be created. The agenda consists of recommender, item, user attribute and user attribute value that are needed to create a micro-task. The agenda, furthermore, has a property *Quantity*. This property indicates how many identical micro-tasks are generated from this agenda. Although, agenda and micro-task include recommender, item, user attribute and user attribute values, they are two different concepts:

- An agenda is an *instruction* for the micro-task scheduler to generate a number of micro-tasks.
- The agenda specifies which recommender, item, user attribute and user attribute value are considered in the micro-task.

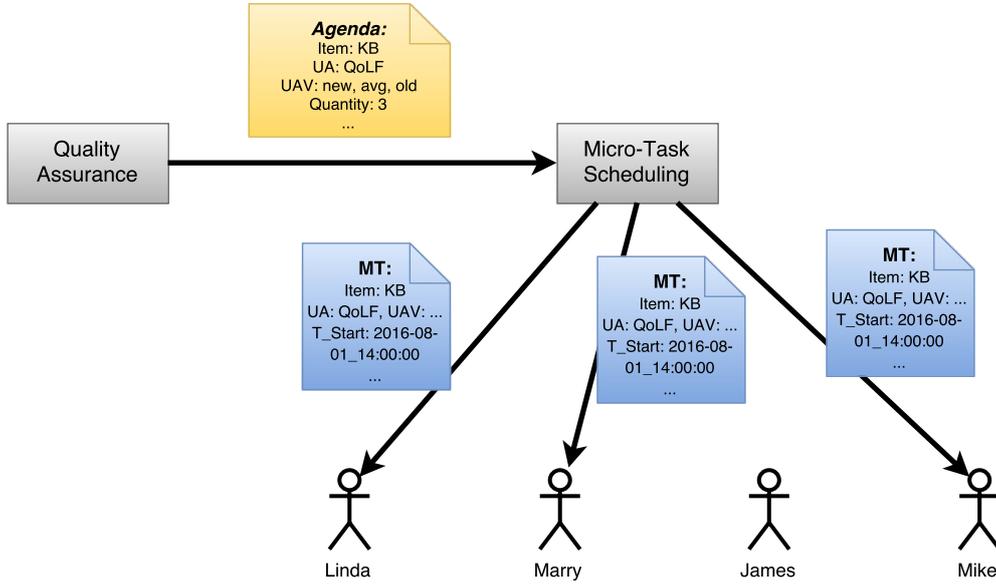


Figure 5.1: Overview of the micro-task generation process based on an agenda.

- It tells the micro-task scheduler how many (given by *Quantity*) micro-tasks need to be generated in which time period (between t_{start} and t_{max}).
- A micro-task includes the provided recommender, item, user attribute and user attribute value. In addition, the micro-task type, the user the task was assigned to and after the task was solved, the finishing date are contained by a micro-task.
- Micro-tasks are associated with a user interface depending on the micro-task type, whereas the agenda is used to pass information from the Quality Assurance to the micro-task scheduler.

Table 5.8 gives an example of an agenda.

Property	Value
t_{start}	01.08.2016 14:00
t_{max}	01.09.2016 14:00
user attribute	Quality of lift facilities
user attribute values	new, average, old
item	Kitzbühel
recommender	Skiing Resorts
Quantity	3
Priority	3

Table 5.8: Example agenda for recommender Skiing Resorts.

Selecting the Type of the Micro-Task based on the Agenda

Micro-tasks generally are created based on the properties specified in the agenda. Different micro-tasks need different information when they are created. As each micro-task type needs other information, a decision tree (see Figure 5.2) can be used to decide which micro-task type results from a given agenda.

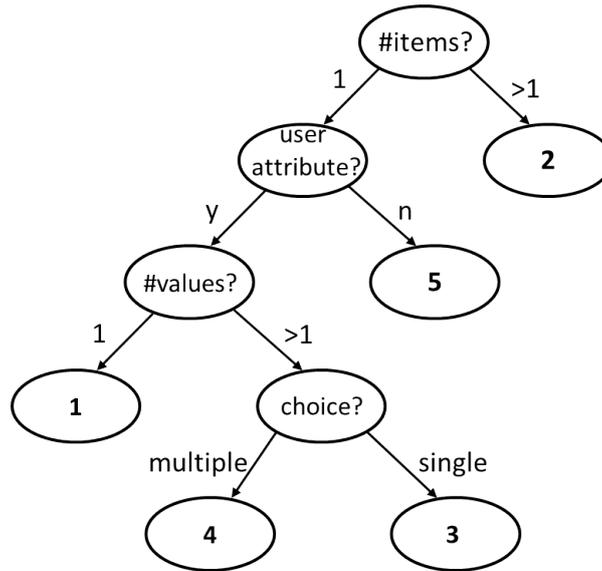


Figure 5.2: The decision tree that is used to determine the micro-task type by the given agenda.

The leafs of the decision tree indicate micro-task types. The different micro-task types and their visualizations were already discussed in Section 4. The following list explains the inner nodes in the decision tree:

- **#items?:** A distinction between micro-tasks showing one item and micro-tasks showing multiple items is taken here. Currently, only one micro-task type contains multiple items (micro-task type 2).
- **user attribute?:** If no user attribute is specified in the agenda, micro-tasks of type 5 are created. Otherwise, further decisions need to be made.
- **#values?:** If only one user attribute value is specified in the agenda, micro-tasks of type 1 are generated. Otherwise, further decisions are necessary.
- **choice?:** Depending on the choice type of the specified user attribute (multiple choice or single choice) either micro-tasks of type 4 or 3 are generated. The choice type of the user attribute is defined by the user when adding the user attribute.

All micro-tasks but type 6 micro-tasks are created from the decision tree. These type 6 micro-tasks are CAPTCHAs and do not rely on the informations from an

agenda. The CAPTCHAs are created based on the human score of a person. The quality assurance contains mechanisms that detect whether a user acts like a human or not. Therefore, a *human score* (see Section 4.5) exists in the system to describe whether the user acts like a human. The value ranges between 0 and 1 where 1 means the user is human and 0 means the user acts non-human. The number of CAPTCHAs distributed to a user is based on this human score. If the human score is high, the user will get a small number of CAPTCHAs, if the human score is small, she will get many CAPTCHAs.

To construct micro-tasks from the agenda in Table 5.8, the following steps using the decision tree are performed:

- There is exactly one item given in the agenda, therefore, the agenda is passed on left path to the next node
- There is at least one user attribute present in the agenda, therefore, again the left path can be used
- More than 1 user attribute values are present in the agenda, therefore, the agenda is passed to the node on the right path
- The choice type of the user attribute needs to be checked. As the user attribute *Quality of lift facilities* allows only single choice, the agenda is passed on the right path.
- The leaf of the decision tree is reached and therefore, the type of the micro-task is known. It is micro-task type 3.

The visualization of the selection process is depicted in Figure 5.3. As the agenda specified a quantity of three, three identical micro-tasks are generated. The only difference between these three micro-tasks can be observed after the micro-task scheduling process as each micro-task will be assigned to a different user. The resulting micro-task before being scheduled can be seen in Table 5.9. The associated view can be seen in Figure 5.4.

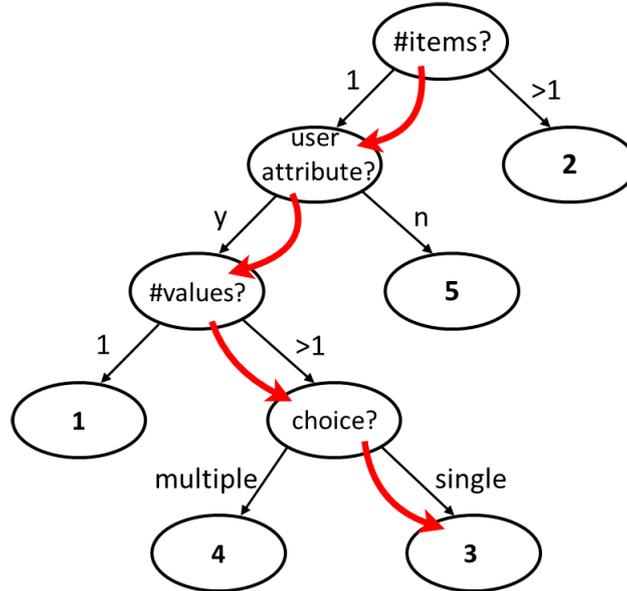


Figure 5.3: Path taken through the decision tree when processing the example agenda shown in Table 5.8.

Property	Value
t_{start}	01.08.2016 14:00
t_{max}	01.09.2016 14:00
t_{finish}	-
user attribute	Quality of lift facilities
user attribute values	new, average, old
item	Kitzbühel
keywords	most legendary winter sports towns especially skiing alps measured top hahnenkamm streif mausefalle challenging pistes perfect children families large continuous skiing area snowboarding mountains snow sledging funpark tirol
recommender	Skiing Resorts
micro-task type	3
user	-

Table 5.9: Micro-task generated from the agenda shown in Table 5.8 before scheduling.

Skiing Resorts



Item »Kitzbühel«: Which answer fits the attribute »Quality of lift facilities« best?

new average

old

How well? ? 0% 100%

Don't show questions for this recommender

Figure 5.4: Associating view of the micro-task created from the agenda in Table 5.8.

5.4 Scheduling Approach

The scheduling of micro-tasks is done based on a score that is computed for each user-micro-task pair. It tries to assign micro-tasks to users which are assumed to be best suited to solve the task. The suitability of users will be calculated and expressed by a function named *score*. The computation of this score takes many aspects of user's interactions with the system as well as some aspects of a micro-task itself into account. In this section, all formulas will be defined and afterwards an illustrative example showing the functionality of the set of formulas will be given based on the previously defined example recommender, items, and users.

The logarithm is used in the entire set of formulas. The reason for this is that originally all formulas were multiplied with each other and not added to each other to compute the score. All the values of the quotients of the formulas range between 0 and 1. If values smaller than 1 are multiplied with each other, they will get even smaller and at some point, the precision of a computer is not enough anymore to deal with such small values. Therefore, the logarithm is used throughout the whole set of formulas to overcome this problem. With the usage of the logarithm, all multiplications are transformed to additions because of the calculation rules for the logarithm. The logarithm does not change the similarity measure between values which means that if one value is large and the other small, they will have the same relation to each other after applying the logarithm to both of them.

Some of the formulas contain weighting parameters ($\alpha_i, \alpha_{e_1}, \alpha_{e_2}, \alpha_w, \alpha_{e_3}$). The weighting factors for all examples are set to one. The weighting factors express the influence of each formula on the *score*. The weighting factors can be tweaked by hand or using a machine learning approach such as a genetic algorithm. To apply a genetic algorithm a certain amount of support values needs to be available, otherwise the learning process would lead to overfitting [47]. The utility function used in the genetic algorithm can be defined as *number of correctly classified items* which is the same metric as used in the evaluation (see Section 6).

All fractions include the addition of one in the denominator to avoid possible divisions by zero.

5.4.1 Definitions

In Equation 5.9, the formula for the computation of the *score* is shown. It depends on a *user* u and a *micro-task* m and is a combination of the *importance* of a *micro-task* m and the *qualification* of a *user* u to perform *micro-task* m . The score gives indication about how high the probability is assumed to be that the user will give a proper answer to this micro-task. The value range of the *score* is always a small negative value. The reason for this is that in almost every equation the logarithm is used on a number smaller than 1. The logarithm of such a number is a small negative number, therefore, the score is also negative. Most of the time it ranges between 0 and -5. The larger (nearer to zero) the score is, the better the user fits

the micro-task.

$$score(u, m) = importance(m) + qualification(u, m) \quad (5.9)$$

Equation 5.10 gives the definition of the *importance* of the *micro-task* m that should be scheduled. The *importance* indicates how urgent the completion of this micro-task is. Each *micro-task* m that was generated from an *agenda* has a due date $t_{max}(m)$ and generation date $t_{start}(m)$. The value $t_{current}(m)$ is the time stamp when the micro-task enters the scheduling process. All timestamps are measured in milliseconds.

$$importance(m) = \log \left(\frac{t_{current}(m) - t_{start}(m) + \alpha_i}{t_{max}(m) - t_{start}(m) + 1} \right) \quad (5.10)$$

The *qualification* of a *user* u to solve a *micro-task* m is defined in Equation 5.11. The *qualification* is composed of the *ability* of a user to solve the micro-task and the *interest* of a user in this micro-task. This formula shows how probable it is that the user can give a good answer for the micro-task and how much she might be interested in such a micro-task.

$$qualification(u, m) = ability(u, m) + interest(u, m) \quad (5.11)$$

In Equation 5.12, the *ability* of a *user* u to solve the *micro-task* m is given. The *ability* indicates the quality of a user's interactions with the system and how good she would be suited to solve the micro-task. It is made up by the *experience* and the *excellence* of a user in the respective recommender.

$$ability(u, m) = experience(u, r(m)) + excellence(u, r(m)) \quad (5.12)$$

The computation of the *experience* is shown in Equation 5.13. The previously defined micro-task and user keywords are used here. The *experience* declares how good the user keywords match the micro-task keywords. Each value $weight(k_m)$ in the numerator is the sum of the user keywords weight and the micro-task keywords weight for all matching user and micro-task keywords. Each value $weight(k)$ in the denominator is the total sum of all user keywords and all micro-task keywords regardless whether they match or not.

$$experience(u, m) = \log \left(\frac{\alpha_{e1} + \sum_{k_m \in kw(u) \cap kw(m)} weight(k_m)}{1 + \sum_{k \in kw(u) \cup kw(m)} weight(k)} \right) \quad (5.13)$$

Equation 5.14 gives the computation rule for the *excellence* of a *user* u in the *recommender* $r(m)$ the micro-task belongs to. The formula indicates the quality of a user's evaluations compared to the evaluations given by the rest of the community. The value $\#correctanswers(u, r(m))$ is the number of correct answers given by the *user* u in the *recommender* r . A correct answer is determined by comparing the support values of the *user* u given for evaluations in this *recommender* $r(m)$ with the aggregated supports for the evaluations given by the community for this recommender. If the difference between the users estimated support and the aggregated support does not exceed a certain value, the answer is treated as correct, otherwise, it is treated as wrong. The threshold is defined by the Quality Assurance and can change with time. The value $\#answers(u, r(m))$ is the total number of support values given by the *user* u for items in the *recommender* $r(m)$.

$$excellence(u, r(m)) = \log \left(\frac{\alpha_{e_2} + \#correctanswers(u, r(m))}{1 + \#answers(u, r(m))} \right) \quad (5.14)$$

The *interest* of a *user* u in a *micro-task* m is defined in Equation 5.15. It is a combination of the *workload* of a *user* u within a specific *time period* Δ and the *emphasis* of the *user* u for the *recommender* $r(m)$. It determines the general willingness of the user to solve micro-tasks in the recommender the micro-task belongs to.

$$interest(u, m, \Delta) = workload(u, \Delta) + emphasis(u, r(m)) \quad (5.15)$$

In Equation 5.16, the *workload* of a *user* u is defined. The workload uses the *time period* Δ to determine how frequently the user interacts with the system, especially evaluates items. The time period Δ is given in days. The value $\#answers(u, \Delta)$ is the number of support values the user entered within the *time period* Δ no matter what recommender the evaluation belonged to. The *time period* Δ in PEOPLEVIEWS implementation is set to 7 days. As the workload does not include a fraction, it needs to be multiplied with minus 1. The reason for that is that a higher workload needs to be indicated by a negative number farther away from zero than a lower workload (workload of zero would be the best case).

$$workload(u, \Delta) = -\log(\alpha_w + \#answers(u, \Delta)) \quad (5.16)$$

Finally, the *emphasis* is defined in Equation 5.17. It represents the general willingness of the user to contribute to the *recommender* $r(m)$ the micro-task belongs to compared to the general contributions to the system. The value $\#answers(u, r(m))$ represents the number of support values in evaluations specified in the recommender $r(m)$. $\#answers(u)$ is the total number of support values given in evaluations over

all recommenders.

$$emphasis(u, r(m)) = \log \left(\frac{\alpha_{e3} + \#answers(u, r(m))}{1 + \#answers(u)} \right) \quad (5.17)$$

After the *score* for each user was calculated, the micro-task is distributed to n best matching users. n is given by the agenda (property *Quantity*) that is generated by the quality assurance. For example, the agenda shown in Table 5.8 has a quantity of $n = 3$.

5.4.2 Example

Reminder and Preparations

As a reminder, the important parts of the example recommenders already described and created in Chapter 3 are shown again.

The users interacting with the system and their corresponding keywords can be seen in Table 5.11. In Table 5.10 the two example recommenders with their keywords are shown. Table 5.12 and 5.13 contain the evaluations done by the users.

Recommender Name	Keywords
Mobile Phones	want buy new mobile need performance need high battery life recommender will help make decision
Skiing Resorts	like skiing snowboarding love winter mountains help find perfect resort

Table 5.10: Examples of recommenders and their keywords extracted from the description.

User	Keywords
Mike	$\text{kw}(\text{Samsung Galaxy S7}) \cup \text{kw}(\text{Apple iPhone 6S}) \cup \text{kw}(\text{Mobile Phones})$
James	$\text{kw}(\text{Samsung Galaxy S7}) \cup \text{kw}(\text{Sony Xperia M5}) \cup \text{kw}(\text{Mobile Phones}) \cup \text{kw}(\text{Kitzbühel}) \cup \text{kw}(\text{Schladming}) \cup \text{kw}(\text{Skiing Resorts})$
Linda	$\text{kw}(\text{Apple iPhone 6S}) \cup \text{kw}(\text{Mobile Phones}) \cup \text{kw}(\text{Schladming}) \cup \text{kw}(\text{Skiing Resorts})$
Marry	$\text{kw}(\text{Sony Xperia M5}) \cup \text{kw}(\text{Mobile Phones}) \cup \text{kw}(\text{Kitzbühel}) \cup \text{kw}(\text{Obertauern}) \cup \text{kw}(\text{Skiing Resorts})$

Table 5.11: User keywords based on their interactions from Table 5.12 and 5.13 and the keywords $\text{kw}(\dots)$ from Table 5.10 and 5.14.

The first step is to calculate the user keywords and item keywords and their weightings as shown in Section 5.2. For the sake of simplicity and due to space limitations there will only be a table containing the user and item keywords without weightings. For an example on the calculation of keywords' weights see Section 5.2. Table 5.14 contains the items and their keywords and Table 5.11 contains the user keywords calculated from the given evaluations.

To begin the example, let's assume there was a new item added to the recommender *Skiing Resorts*. The item *Nassfeld* can be seen in Table 5.15.

The keywords for the item *Nassfeld* are:

nassfeld one top ski resorts austria excited perfectly groomed slopes unique mountain panorama plenty sun special services lifts gondolas six person chairlifts quad

User	Item	Performance	Value for money	Battery life for different usages	Design
Mike	Samsung Galaxy S7	acceptable (80%)	looses slightly (60%)	gaming(80%) browsing(75%) photo(70%)	valuable (80%)
James	Samsung Galaxy S7	very good (75%)	looses slightly (80%)	photo(70%) gaming(85%)	moderate (90%)
Linda	Apple iPhone 6S	very good (90%)	keeps value (70%)	photo(80%) browsing(75%)	valuable (80%)
James	Sony Xperia M5	acceptable (60%)	looses fast (60%)	telephoning(80%) browsing(70%)	moderate (70%)
Marry	Sony Xperia M5	very good (75%)	looses slightly (80%)	browsing(85%) photo(75%)	valuable (90%)
Mike	Apple iPhone 6S	excellent (80%)	looses slightly (75%)	gaming(80%) browsing(70%) photo(60%)	moderate (80%)

Table 5.12: Example evaluations of items of the recommender Mobile Phones. The support is given as a percentage value in brackets.

chairlifts drag lifts magic carpets perfectly groomed slopes difficult medium easy austria longest floodlit slopes alps exclusive freeride funareas snowpark long valley run guaranteed snow snow making sunshine express ski hotspots kids practice area families

Further assumptions to be made are the number of evaluations done by the users within a time period, the number of correct evaluations in both recommenders, and the total number of evaluations in both recommenders. Table 5.16 gives assumptions for those numbers.

Finally, to generate micro-tasks, an agenda is needed. The quality assurance detected the newly added item *Nassfeld* in the database and therefore, created the agenda shown in Table 5.17. This agenda will be used to generate the micro-tasks from.

User	Item	Price level	Experience level needed	Quality of lift facilities	Target audience	Quality of ski lodges
Marry	Kitzbühel	exp. (80%)	moderate (90%)	average (70%)	rec. athlete (70%) snowboarder (80%)	good (70%)
James	Kitzbühel	mod. (90%)	high (75%)	new (80%)	top. athlete (70%) rec. athlete (70%)	moderate (80%)
James	Schladming	mod. (80%)	moderate (75%)	new (70%)	rec. athlete (80%) rec. family (90%)	good (80%)
Linda	Schladming	mod. (70%)	low (70%)	average (90%)	snowboarder (80%) family (80%)	good (90%)
Marry	Obertauern	cheap (70%)	moderate (65%)	new (80%)	rec. athlete (70%) family (65%)	moderate (75%)

Table 5.13: Example evaluations of items of the recommender Skiing Resorts. The support is given as a percentage value in brackets.

Item	Keywords
Kitzbühel	most legendary winter sports towns especially skiing alps measured top hahnenkamm streif mausefalle challenging pistes perfect children families large continuous skiing area snowboarding mountains snow sledging funpark tirol
Schladming	mountains skiing area ski mountain shows best one thing common offer fine slopes modern cable cars lifts culinary delights huts offers kids families athletes connoisseurs styria funpark families snowboarding skiing snow
Obertauern	one snowiest ski areas alps means perfect conditions late november early may middle salzburg mountains gondola chairlifts drag lifts fantastic ski runs cross country trails plenty excellent après ski nightlife modern hotel standards top wellness fitness offer wonderfully carefree holiday mood funpark snowkiting skiing beautiful kombibahn
Samsung Galaxy S7	features fine interplay glass metal large quad hd display super amoled technology ensures brilliant images consistently sharp writing driven galaxy octa core processor memory side stand networking functionality combines bandwidth wlan mimo lte faster downloads shorter buffer load times hd video streaming calling complex sites megapixel camera offers fast autofocus focuses seconds display performance video security samsung galaxy
Sony Xperia M5	get chance get perfect shot camera xperia packed full technology help capture moment shoot pro hybrid autofocus megapixels super sharp zoom camera display performance storage camera sony
Apple iPhone 6S	inch retina hd display touch aluminum stronger glass cover chip desktop architecture brand new megapixel isight camera live pictures touch id faster lte wlan long battery life iOS iCloud smooth continuous unibody design live pictures camera display retina iphone apple

Table 5.14: Item keywords for the recommenders Skiing Resorts and Mobile Phones.

Item Name	Family Friendly	Sledging	Cross-country skiing	Kilometers of slopes	types of lift facilities
Nassfeld	Yes	Yes	Yes	110	cableway, chair lift, drag lift

Table 5.15: Information about the new item Nassfeld of the recommender Skiing Resorts.

User	Recommender	#correctanswers	#answers	#answers(Δ)
Mike	Mobile Phones	7	12	6
James	Mobile Phones	6	10	5
James	Skiing Resorts	8	12	6
Linda	Mobile Phones	4	5	0
Linda	Skiing Resorts	3	6	2
Marry	Mobile Phones	3	5	2
Marry	Skiing Resorts	5	6	3

Table 5.16: Needed assumptions for each recommender for the number of correct support values given, the total number of support values given, and the number of support values given within a time period Δ .

Property	Value
t_{start}	28.05.2016 16:23
t_{max}	15.06.2016 16:23
user attribute	Target Audience
user attribute values	family, recreational athlete, top-class athlete, snowboarder
item	Nassfeld
recommender	Skiing Resorts
Quantity	2
Priority	3

Table 5.17: Example agenda for recommender Skiing Resorts of item Nassfeld.

Calculations

From the agenda shown in Table 5.17, a *micro-task* m_1 of type 4 related to the item *Nassfeld* and user attribute *Target Audience* belonging to recommender *Skiing Resorts* needs to be created. The type again is determined using the decision tree depicted in Figure 5.2. The steps taken in the decision tree according to the agenda in Table 5.17 can be seen in Figure 5.5. The agenda contains exactly one item and a user attribute. It also contains more than one user attribute value and the choice type for the user attribute is multiple choice. These properties lead to micro-tasks of type 4. The given quantity in the agenda is 2, therefore, 2 micro-tasks need to be generated and distributed.

Generally, larger values (closer to zero) of all calculated (intermediate) results indicate a better fit of the user for the respective formula than a value father away from zero.

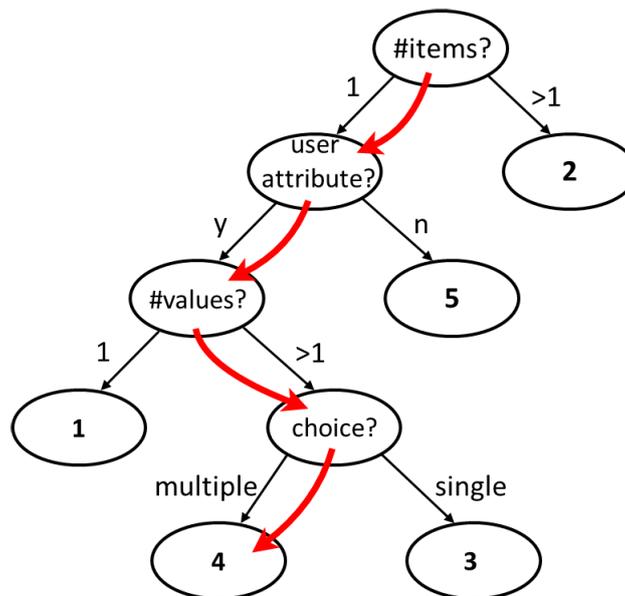


Figure 5.5: Path taken through the decision tree when processing the example agenda shown in Table 5.17.

First of all, the importance of the micro-task is calculated in Equation 5.18. The time stamps are given in milliseconds. Start time $t_{start}(m_1)$ is May 28 2016 16:23, current time $t_{current}(m_1)$ is May 29 2016 16:23 and due time $t_{max}(m_1)$ is June 15 2016

Property	Value
t_{start}	28.05.2016 16:23
t_{max}	15.06.2016 16:23
t_{finish}	-
user attribute	Target Audience
user attribute values	family, recreational athlete, top-class athlete snowboarder
item	Nassfeld
keywords	nassfeld one top ski resorts austria excited perfectly groomed slopes unique mountain panorama plenty sun special services lifts gondolas six person chairlifts quad chairlifts drag lifts magic carpets perfectly groomed slopes difficult medium easy austria longest floodlit slopes alps exclusive freeride funareas snowpark long valley run guaranteed snow snow making sunshine express ski hotspots kids practice area families
recommender	Skiing Resorts
micro-task type	4
user	-

Table 5.18: Micro-task resulting from the agenda shown in Table 5.17.

16:23. As already stated, the time values in the equation are given in milliseconds.

$$\begin{aligned}
importance(m_1) &= \log \left(\frac{t_{current}(m_1) - t_{start}(m_1) + \alpha_i}{t_{max}(m_1) - t_{start}(m_1) + 1} \right) \\
&= \log \left(\frac{1464531780000 - 1464445380000 + 1}{1466000580000 - 1464445380000 + 1} \right) \quad (5.18) \\
&= -1.25527
\end{aligned}$$

The *interest* of the users in this micro-task is calculated. Therefore the *workload* and *emphasis* of each user must be calculated. Table 5.19 contains the interest of the users in the micro-task and also the corresponding *workload* and *emphasis* values.

User	Interest	Workload	Emphasis
Mike	-1,95904	-0.84510	-1,11394
James	-1,32696	-1.07918	-0,24778
Linda	-0,71120	-0.47712	-0,23408
Marry	-1,01223	-0.77815	-0,23408

Table 5.19: Interest, workload and emphasis of all users in the system. Values closer to zero indicate a higher interest, lower workload, and higher emphasis respectively.

As an example, the *workload*, *emphasis*, and *interest* of the user *James* will be calculated here. Equation 5.19 shows the computation of the workload. As already mentioned, the time period Δ is 7 days in this example. The total number of answers of user *James* is the sum of the number of answers in all recommenders in the time period Δ . In this example, the total number is 11, as for *Skiing Resorts* it is 6 and for *Mobile Phones* it is 5 as mentioned in Table 5.16 and they are summed up.

$$\begin{aligned} workload(James) &= -\log(\alpha_w + \#answers(James, \Delta)) \\ &= -\log(1 + 11) = -1.07918 \end{aligned} \quad (5.19)$$

The emphasis of user *James* is computed in Equation 5.20. *Skiing Resorts* is abbreviated as *SR* in the formula to keep it shorter.

$$\begin{aligned} emphasis(James, SR) &= \log\left(\frac{\alpha_{e_3} + \#answers(James, SR)}{1 + \#answers(James)}\right) \\ &= \log\left(\frac{1 + 12}{1 + 22}\right) = -0,24778 \end{aligned} \quad (5.20)$$

Now that *workload* and *emphasis* have been computed, the *interest* of user *James* is computed in Equation 5.21.

$$\begin{aligned} interest(James, m_1, 7) &= workload(James, 7) + emphasis(James, SR) \\ &= -1.07918 + (-0,24778) = -1,32696 \end{aligned} \quad (5.21)$$

The next value to compute is the *ability* of the users to solve the micro-task. To calculate the *ability*, *experience* and *excellence* need to be computed first. Table 5.20 shows the calculated values for *ability*, *experience*, and *excellence* for each user.

User	Ability	Experience	Excellence
Mike	-1,40324	-1,40324	0
James	-0,65099	-0,49129	-0,15970
Linda	-0,95462	-0,71158	-0,24304
Marry	-0,46883	-0,40188	-0,06695

Table 5.20: Ability, experience and excellence of this micro-task of all users in the system. Values closer to zero indicate a higher ability, higher experience and higher excellence respectively.

To compute the *experience*, the user and micro-task keywords and their weights are needed. As the computation of weights has already been shown (see Section 5.2.2) no calculation examples will be done here.

The experience of user *James* is calculated in Equation 5.22. The calculation of the summing is not shown in detail as the sums involve too much summands (see Table 5.11 for keywords of user *James* and keywords for *Nassfeld* in Section 5.4.2; 223 keywords in total).

$$\begin{aligned}
experience(James, m_1) &= \log \left(\frac{\alpha_{e_1} + \sum_{k_m \in kw(James) \cap kw(m_1)} weight(k_m)}{1 + \sum_{k \in kw(James) \cup kw(m_1)} weight(k)} \right) \\
&= \log \left(\frac{1 + weight(snow) + \dots + weight(mountain)}{1 + weight(features) + \dots + weight(families)} \right) \\
&= \log \left(\frac{1 + 35.74361}{1 + 112.88743} \right) = -0,49129
\end{aligned} \tag{5.22}$$

Now the excellence of user *James* needs to be computed. This can be seen in Equation 5.23.

$$\begin{aligned}
excellence(James, SR) &= \log \left(\frac{\alpha_{e_2} + \#correctanswers(James, SR)}{1 + \#answers(James, SR)} \right) \\
&= \log \left(\frac{1 + 8}{1 + 12} \right) = -0,15970
\end{aligned} \tag{5.23}$$

After *experience* and *excellence* were calculated, the *ability* of user *James* can be calculated as shown in Equation 5.24.

$$\begin{aligned}
ability(James, m_1) &= experience(James, SR) + excellence(James, SR) \\
&= -0.49129 + (-0.15970) = -0,65099
\end{aligned} \tag{5.24}$$

Using the previously calculated values for *ability* and *interest*, the *qualification* of each user can be calculated. Furthermore the total *score* can be calculated as the *importance* was already computed. Table 5.21 shows the final *score*, *qualification*, *ability*, *interest*, and *importance* for the users and the micro-task.

The calculation of the qualification is shown in Equation 5.25.

$$\begin{aligned}
qualification(James, m_1) &= ability(James, m_1) + interest(James, m_1) \\
&= -0,65099 + (-1.32696) = -1,97795
\end{aligned} \tag{5.25}$$

The final *score* is computed in Equation 5.26.

$$\begin{aligned}
score(James, m_1) &= importance(m_1) + qualification(James, m_1) \\
&= -1.25527 + (-1,97795) = -3,23322
\end{aligned} \tag{5.26}$$

User	Score	Qualification	Ability	Interest	Importance
Mike	-4,61755	-3,36228	-1,40324	-1.95904	-1.25527
James	-3,23322	-1,97795	-0,65099	-1.32696	-1.25527
Linda	-2,92109	-1,66582	-0,95462	-0.71120	-1.25527
Marry	-2,73633	-1,48106	-0,46883	-1.01223	-1.25527

Table 5.21: Final scores of each user and the necessary values to compute the score. Best suiting users are selected based on their score. Users are assumed to be most suitable for micro-tasks if the score is closer to zero.

The final results in Table 5.21 show that the 2 best suited users to complete the micro-task are *Marry* and *Linda* as their *scores* are the largest (closest to zero). Therefore, *Marry* and *Linda* each get assigned the micro-task shown in Table 5.18.

6

Evaluation

This chapter contains the evaluation for the scheduling approach described in Chapter 5. The baseline algorithms, the scheduling approach was compared with, are also discussed in this chapter. Furthermore, the used dataset will be described and also the evaluation results are discussed in detail. The problems when evaluating the algorithm as well as the limitations due to the dataset and the solutions to the occurred problems are illustrated.

6.1 Dataset

To evaluate the scheduling approach described in Chapter 5, no dataset exists on which all the presented formulas can be directly applied. The requirements regarding such a dataset would be:

1. A large amount of users (at least thousand) is necessary for the study to be representative and show that the scheduling approach is scaling to a large user base.
2. A large amount of items (at least thousand) is needed such that separate training and tests set can be extracted.
3. Detailed, objective, textual item descriptions are needed to extract keywords for the items and the user profile.
4. History of user interactions with items is necessary to create the user profile and estimate the user's interests. To create a representative user profile a certain amount of user interactions needs to be present (at least 10 interactions in a certain domain could indicate interest in that domain).

5. A high amount of evaluations per item is needed to estimate average ratings and to find diverging user evaluations. This information is used to estimate a user's experience in a certain recommender domain.
6. Different recommender domains are necessary because the approach includes user's interest in different domains.

For the existing datasets collected using PEOPLEVIEWS (collected for the evaluation done in the master's thesis of Michael Schwarz) requirements 2, 4, and 5 can not be met. In those studies only a small set of items was used (less than 15 items per recommender domain). Also users were asked to evaluate exactly three items which is insufficient to determine users interests. Furthermore, one of the two existing recommender domains in the study was assigned randomly to study participants. As study participants were paid for completing the study, users only used the system once to complete the study without using the system again. As noted by Schnitzer et al. [28], paid micro-workers tend to maximize their revenue per time unit, therefore, their behavior when using the system is different to normal users.

To collect an appropriate dataset a long-term study would be necessary. The setup of such a study will be described in Chapter 7.

The dataset that fulfilled the most requirements was the *MovieLens 10M* data set [48]. To meet all specified requirements some adaptations to the data set were necessary which will be described after introducing the *MovieLens* dataset. The *MovieLens* dataset contains 10 000 054 ratings, 95 580 tags applied to 10 681 movies and 71 567 users. The data set is provided by the online movie recommendation system *MovieLens*. This data set was the only one containing some kind of description in form of genres and tags. Tags are not only one single word but can be a whole sentence.

After analyzing the tags, it was clear that they are not suitable to be used for the study as they were very subjective to users opinions instead of objective movie descriptions. As it turned out, users misused tags to *textually rate* the movies. Some of the most used tags include *classic, excellent, funny, bad movie, don't watch it*. As clearly can be seen such tags do not provide any objective description of a movie. Therefore, detailed movie descriptions needed to be supplemented to the *MovieLens* dataset.

The *Netflix Roulette* database [49] is a web service providing detailed informations about movies such as movie descriptions or actors. To get a movie description, the function *getAllData(mn)* from the provided API is used. The argument *mn* represents the *name* of the movie. As this database does not contain ratings, the two data sets *MovieLens* and *Netflix Roulette* need to be combined. The *Netflix Roulette* database provides the needed movie description and the *MovieLens* data set contains the users, movies, and ratings.

Another problem is that the ratings are just star ratings in *MovieLens*. Star ratings are often not subjective or biased for some reason. For example, a movie

can get a bad rating because the user does not like one of the actors. Because of this, the value of the star rating is not taken into account in the evaluation; just the fact that the user actually evaluated the movie or not.

While trying to get the movie descriptions, a problem occurred. When using the movie names as arguments, not all movies were found in the *Netflix Roulette* database. As the function of the API just takes the name of the movie, the problem might have occurred because the movie name in the *MovieLens* database was written differently than in the *Netflix Roulette* database. This is a problem that could not be solved, as there was no other way to get the needed information out of the database. This resulted in a reduction of the final data set size to 2016 movies, 6 040 532 ratings and 58 126 users. Users that did not evaluate any remaining movie were removed from the dataset. Nevertheless, the combined dataset still met all specified requirements.

Simplifications of the Scheduling Algorithm

Due to the differences between the *MovieLens* dataset and the PEOPLEVIEWS structure, some simplifications were applied to be able to use the formulas of the *scheduling* algorithm. For each component of the formulas presented in Section 5.4 the used data sources and eventually simulated data are listed. No modifications to the formulas were made.

- The *importance* (Equation 5.10) could not be calculated as there are no time restrictions for micro-tasks available in the *MovieLens* dataset. As it is not possible to estimate or simulate such information, the *importance* was set to 1 for all micro-tasks.
- The *experience* (Equation 5.13) can be calculated using the keywords extracted from the movie descriptions which were fetched from *Netflix Roulette*. The history of evaluations was used to generate the user profiles.
- The *excellence* (Equation 5.14) values were simulated. To simulate them, first the average rating for each movie was calculated. Evaluations which were within ± 0.5 of this average were deemed as correct answers.
- The *workload* (Equation 5.16) was calculated using the history of users evaluations.
- The *emphasis* (Equation 5.17) was calculated by treating the movie genres as different recommender domains.
- All weighting parameters ($\alpha_i, \alpha_{e_1}, \alpha_{e_2}, \alpha_w, \alpha_{e_3}$) were set to 1. If they would have been optimized by a genetic algorithm, the problem of overfitting could have occurred which would have falsified the results of the *scheduling* algorithm.

6.2 Evaluation Approach

Before describing the evaluation approach, the algorithms, the scheduling approach was compared to, will be described. The three baseline algorithms used are *item-similarity* (case-based), *Most Frequent User*, and *Random User*. Item-similarity, Most Frequent User, and Random User were chosen because those are algorithms are applicable to the chosen dataset and are widely used as baselines in research studies.

In this section, several abbreviations are used. Table 6.1 contains each abbreviation, a short description and the scope of the abbreviation. Possible scopes are *item-similarity*, *evaluation algorithm* and *both*. They describe in which context each abbreviation is valid.

Item-similarity

Item-similarity or case-based recommendation [50] is a form of knowledge-based recommendation and performs very well in domains such as books or digital cameras where products are defined in terms of a set of properties such as price or color. Instead of properties, in this evaluation, keywords are matched. With this algorithm it is possible to find users who should evaluate a new movie, that is similar to the movies she rated in the past based on matching keywords. The following steps are performed to find suitable users for new movies:

- Compare the movie keywords from the evaluation movie in m_{ev} with the movie keywords from the already evaluated movies in m_r .
- Add each already evaluated movie in m_r where more than 5 keywords match the keywords from an evaluation movie in m_{ev} to a set l
- From each movie m_e in the set l :
 - Add the users who evaluated this movie to the set l_u
 - Select n users randomly from the list l_u that should evaluate the evaluation movie m_e and store them in u_s .

The threshold of 5 matching keywords was empirically determined as this resulted in the highest percentage of correctly chosen users for the evaluation of the evaluation movie m_e .

The discussed algorithm would be applied to subitem 5(a)i of the evaluation algorithm. Therefore, the selection of m_{ev} and m_r is not described.

Most Frequent User

The *Most Frequent*(MF) algorithm selects the most frequent users from the data set and in this case, suggests them as best matching users to solve the micro-task.

Abbreviation	Description	Scope
M	Total set of movies	both
U	Total set of users	both
R	Total set of ratings	evaluation algorithm
m	Movies randomly selected from M	both
m_{ev}	Evaluation movies randomly selected from m	both
m_r	Remaining movies in m after selecting m_{ev}	both
m_e	Evaluation movie with more than 5 matching keywords	item-similarity
l	Set of movies with more than 5 matching keywords	item-similarity
l_u	Set of users who evaluated the movie m_{ev}	item-similarity
r	Ratings of the movies m	evaluation algorithm
r_r	Subset of r containing all ratings of the movies m_r	evaluation algorithm
r_{ev}	Subset of r containing all ratings of the movies m_{ev}	evaluation algorithm
u_s	Set of users selected by the baseline algorithms or the <i>scheduling</i> algorithm	both
c_c	Number of correctly identified user-movie combinations	evaluation algorithm
c_n	Number of wrongly identified user-movie combinations	evaluation algorithm

Table 6.1: List of abbreviations with descriptions used in either the item-similarity approach, the evaluation algorithm or both.

The most frequent users are the ones who added the most ratings to the data set. The number of ratings a user provided is calculated and then the users are ranked descending by the number of ratings they provided. When used in the evaluation, the first n users are selected from this set and chosen to solve the given micro-task. Table 6.2 contains four randomly chosen users extracted from the *MovieLens* dataset and their corresponding number of ratings. Table 6.3 shows the list after ordering according to the number of ratings. If, for example, a micro-task needs to be distributed to 2 people, they would be assigned to the users u_2 and u_4 according to Table 6.3.

User	Number of Ratings
u1	941
u2	1 639
u3	786
u4	1 306

Table 6.2: Example of the number of ratings for some users before ordering.

User	Number of Ratings
u2	1 639
u4	1 306
u1	941
u3	786

Table 6.3: Table 6.2 after ordering descending by the number of ratings.

Random User

This algorithm is rather easy. For each given micro-task, a specific number of users who should solve the task is selected completely random from the whole set of all users. In contrast to the *Most Frequent User* algorithm where always the same users are chosen for the tasks, here for each micro-task a random set of users is chosen to solve the task.

Evaluation algorithm

The evaluation algorithm predicts which movie should be evaluated by which user. This is equivalent to predicting which micro-task should be solved by which user. Therefore, it is feasible to use such an algorithm to evaluate the micro-task scheduling approach.

The dataset D initially contains the set of movies M , the set of users U and the set of ratings R . Each rating consists of the user u that entered the rating, the movie m that was rated and the rating value r_v . As the goal of the scheduling algorithm is to predict suitable users, the rating value r_v will not be used in the evaluation algorithm. Furthermore, the rating value could not be predicted by the scheduling algorithm, as it is not designed to predict the rating value but the user who should rate a movie. Each step of the evaluation algorithm will be explained in detail in the following list. To clarify the different entries in the used sets, examples will be given in tables referenced in each item. To keep the examples small, 8 movies m and 2 movies m_{ev} are chosen.

1. Randomly choose a subset of movies m (see Table 6.5) from the total set of movies M (see Table 6.4). Only movies in m will be used to train and test the algorithms. The size of m varies to show how the prediction quality changes depending on the size of the dataset.
2. Choose randomly a set m_{ev} (see Table 6.7) as evaluation movies from the set of movies m . For these movies m_{ev} the scheduling algorithm will choose the best suitable users. The remaining set of movies is m_r (see Table 6.6) after removal of the evaluation scheduling movies m_{ev} from m . These m_r movies are used as training data for the scheduling algorithm.

-
3. The set of ratings r (see Table 6.8) regarding all movies in m is split in two subsets. All ratings r_r (see Table 6.9) for the movies m_r are used as training data. The ratings r_{ev} (see Table 6.10) are ratings for the movies in m_{ev} and will be used when testing the algorithm.
 4. Training phase:
 - a) For each movie of the set m_r generate the item keywords.
 - b) Generate user keywords for each user in the set U based on her evaluated items.
 5. Test phase: For each algorithm (Item-similarity, most frequent user, random user, and the micro-task scheduling algorithm):
 - a) For each evaluation movie in m_{ev} :
 - i. Choose the set of suitable users u_s (see Tables 6.11 and 6.12) from the total set of user U by applying the respective algorithm.
 - A. For each rating in r_{ev} where the user who created the rating is contained in u_s , increase the count c_c of correctly matching user-movie combinations.
 - B. If, for a rating in r_{ev} , there exists no matching user in u_s , then increase the count c_n of the non-matching user-movie combinations.
 - b) Finally, compute the total sum over all of correctly matching user-movie combinations c_c and and the total sum over all non-matching user-movie combinations c_n of all movies in m_{ev} .
 - c) With the two total sums, the percentage of correctly identified user-movie pairs can be calculated by dividing the sum of all correctly-matching user-movie combinations by the sum over all correctly-matching and non-matching user movie-combinations. Equation 6.1 shows the calculation of the percentages as plotted in Figure 6.1.

$$\text{Correct classified movies in \%} = \frac{c_c}{c_c + c_n} \cdot 100 \quad (6.1)$$

Movie ID	Movie name
1	Toy Story
2	Cutthroat Island
3	Easy Rider
4	Star Wars: Episode III
5	Batman Begins
6	Sabrina
7	3:10 to Yuma
8	Shoot 'Em up
9	Juno
11	Cloverfield
12	Breakfast at Tiffany's
13	Casablanca
	...

Table 6.4: Extract of the total set of movies M with movie ID and movie name.

Movie ID	Movie name
4	Star Wars: Episode III
5	Batman Begins
9	Juno
12	Breakfast at Tiffany's
43	Superman
76	Fools Rush In
97	Alien
133	Grease

Table 6.5: Set of movies m with movie ID and movie name chosen randomly from M .

Movie ID	Movie name
4	Star Wars: Episode III
5	Batman Begins
9	Juno
12	Breakfast at Tiffany's
76	Fools Rush In
97	Alien

Table 6.6: Set of movies m_r with movie ID and movie name after choosing the movies m_{ev} .

Movie ID	Movie name
43	Superman
133	Grease

Table 6.7: Set of movies m_{ev} with movie ID and movie name chosen randomly from m .

Rating ID	Movie ID	User ID
4	4	u_1
8	4	u_2
8	4	u_3
12	5	u_1
45	5	u_2
65	5	u_5
69	9	u_1
73	9	u_3
81	9	u_4
88	12	u_3
90	12	u_4
92	43	u_1
105	43	u_2
120	43	u_5
134	76	u_1
138	76	u_5
179	97	u_1
190	97	u_2
204	97	u_5
207	133	u_3
209	133	u_4

Table 6.8: List of all ratings in r with rating ID, movie ID and user ID.

User ID
u_1
u_2
u_3
u_5

Table 6.11: Suitable users u_s for evaluation movie Superman. Green colored users actually evaluated the movie Superman whereas red colored users did not.

Rating ID	Movie ID	User ID
4	4	u_1
8	4	u_2
8	4	u_3
12	5	u_1
45	5	u_2
65	5	u_5
69	9	u_1
73	9	u_3
81	9	u_4
88	12	u_3
90	12	u_4
134	76	u_1
138	76	u_5
179	97	u_1
190	97	u_2
204	97	u_5

Table 6.9: Set of ratings r_r of the movies m_r with rating ID, movie ID and user ID.

Rating ID	Movie ID	User ID
92	43	u_1
105	43	u_2
120	43	u_5
207	133	u_3
209	133	u_4

Table 6.10: Set of ratings r_{ev} of the movies m_{ev} with rating ID, movie ID and user ID.

User ID
u_1
u_3
u_4
u_5

Table 6.12: Suitable users u_s for evaluation movie Grease. Green colored users actually evaluated the movie Grease whereas red colored users did not.

The Tables 6.11 and 6.12 were generated using the *item-similarity* algorithm. As it can be seen from Table 6.11, the number of correctly identified movie-user combinations c_c is 3 for movie *Superman* whereas Table 6.12 shows that for movie *Grease* it is 2. The number of wrongly identified user-movie combinations c_n is 1 for movie *Superman* and 2 for movie *Grease*.

With the values for c_c and c_n for both movies, the percentage of correctly classified movies can be calculated by summing up the c_c values and the c_n values and using Equation 6.1. Equation 6.2 shows the percentage calculation for the correctly classified users in this example over all selected evaluation movies.

$$\text{Correctly classified movies in \%} = \frac{c_c}{c_c + c_n} \cdot 100 = \frac{3 + 2}{3 + 2 + 1 + 2} \cdot 100 = 62,5\% \quad (6.2)$$

This evaluation algorithm is executed on the data set several times with different numbers of randomly chosen movies m , randomly chosen evaluation movies m_{ev} and chosen users k . For each parameter set, several runs are executed to get meaningful evaluation results.

In total, the evaluation algorithm was executed with 11 different parameter settings. For each parameter setting, 20 runs were made to calculate an average. The different parameter settings for each set of runs are listed in Table 6.13.

Size of	Setting 1	Setting 2	Setting 3	Setting 4	Setting 5	Setting 6
m	10	20	50	75	100	150
m_{ev}	3	5	10	10	15	20
k	20	30	40	50	55	75

Size of	Setting 7	Setting 8	Setting 9	Setting 10	Setting 11
m	300	600	1000	1500	2016
m_{ev}	25	30	35	40	50
k	100	120	150	200	220

Table 6.13: The 11 different parameter settings used for the evaluation of the scheduling algorithms.

6.3 Evaluation Results

Figure 6.1 shows the evaluation results for the different algorithms. In the plot the x-axis is logarithmic. This is just done to get a better visualization of the results. As

the logarithm does not change the order relation it does not distort the final results. The x-axis represents the increasing number of chosen movies m . The y-axis shows the percentage of correctly matching user-movie combinations.

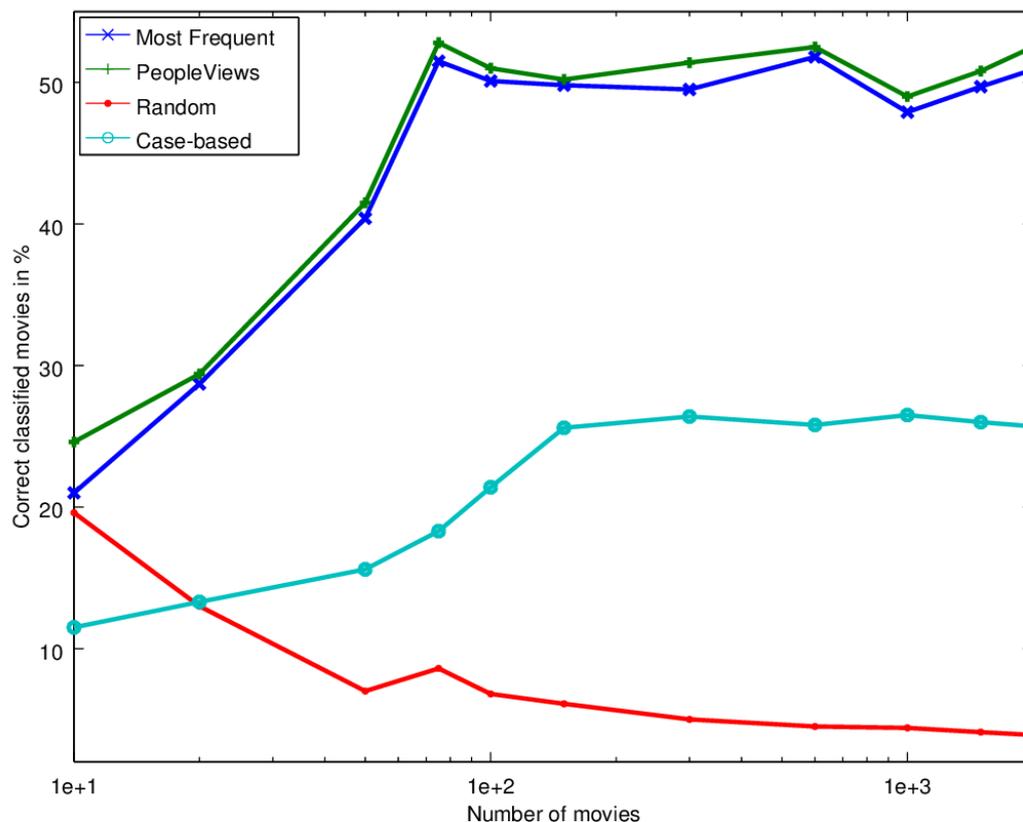


Figure 6.1: The correctly matching user-movie combinations in percent for the scheduling algorithm, the most frequent user algorithm, the case-based recommendation algorithm, and the random user algorithm.

What can also be seen from the plot is that the performance of the *Random User* algorithm decreases steadily with an increasing number of movies. The reason for this is that with the higher number of movies also a higher number of users are taken into account and this leads to a decrease of matches for the *Random User* algorithm. The results reflect the expected behavior of the algorithm. As there are more users, the algorithm can choose from, it is harder to find matching ones as the number of possible users gets very large.

For the *Most Frequent User* and the *scheduling* algorithm the trend of their performance is similar. The *scheduling* algorithm performs for all parameter settings

(see Table 6.13) slightly better than the *Most Frequent User* algorithm. Figure 6.2 shows the *improvement* of the *scheduling* algorithm compared to the *Most Frequent User* algorithm. Equation 6.3 shows the formula for computing the *improvement* of algorithm a_1 compared to algorithm a_2 . The function $c(a)$ gives the *correctly classified movies in %* of algorithm a . A positive improvement value means that a_1 performs better than a_2 , a negative improvement value means that a_1 performs worse than a_2 . The *scheduling* algorithm is always between 1 and 4 percent better than the *Most Frequent User* algorithm except for the beginning, there the *scheduling* algorithm outperforms the *Most Frequent User* algorithm by around 17 percent. In this case, there is a very small number of movies with a small number of users to be distributed to just 3 evaluation movies. This setting simulates some kind of cold start. At the beginning, there is a very small number of existing movies and some ratings, then a new item is added to the data set and correct users need to be chosen. With a large number of data, it is much easier to find a good prediction but with a small one it is harder to make a good prediction. Also with fewer ratings it is hard for the *Most Frequent User* algorithm to make a good prediction as many users have the same amount of ratings. As can be seen, the prediction of the *scheduling* algorithm is apparently much better than for a small number of movies and ratings.

$$Improvement(a_1, a_2) = \frac{c(a_1) - c(a_2)}{c(a_2)} \cdot 100 \quad (6.3)$$

But for a growing data set the *Most Frequent User* algorithm is getting nearly as good as the *scheduling* algorithm on this data set. The reason for this is that for an increasing set of movies also the set of ratings and therefore users increases. As the number of ratings gets larger and larger, the power users are revealed. Power users are users who interact very frequently with the system and therefore make many evaluations. There are users in this data set with over 1500 evaluations. So they evaluated more than 3/4 of all movies. Therefore the algorithm computes such good results. This is also an expected result as it was clear from the beginning that there are many power users in this large data set.

When investigating the waveform of the *item-similarity* approach in Figure 6.1, it can be seen, that the percentage of correctly classified movies increases slower than for the *Most Frequent User* algorithm or the *scheduling* algorithm. It is not really clear why this behavior can be observed but apparently the performance of the *case-based recommendation* approach also increases with the increasing number of movies but not as fast as the *Most Frequent User* algorithm or the *scheduling* algorithm do.

In Figure 6.3, the improvement of the micro-task scheduling algorithm compared to the case-based recommendation algorithm can be seen. It can be seen that in the beginning, the *item-similarity* algorithm performs much worse than the *scheduling*

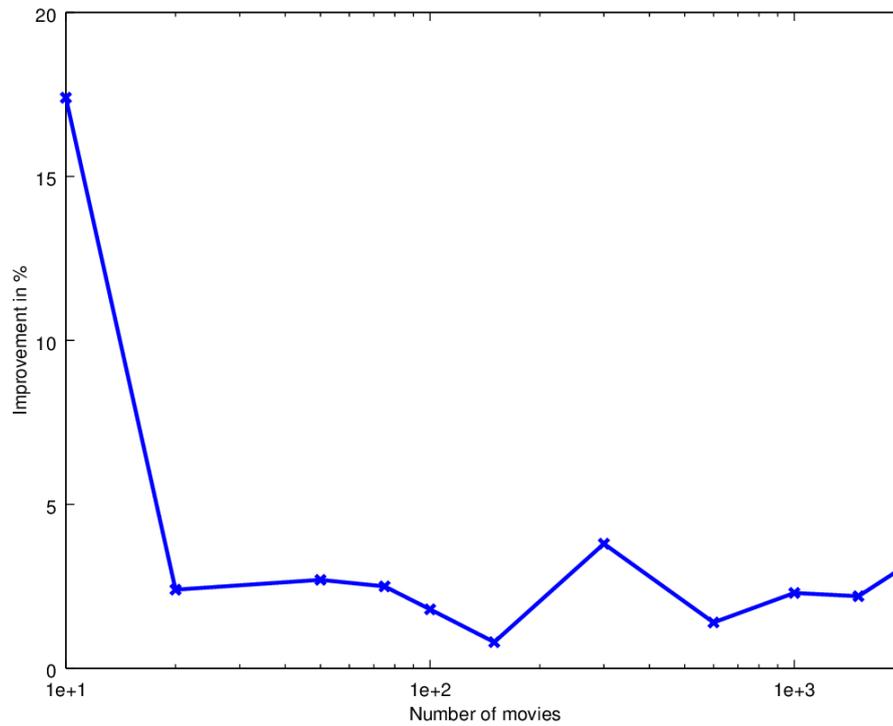


Figure 6.2: The improvement in percent of the scheduling algorithm compared to the Most Frequent User algorithm.

algorithm. As can be seen in Figure 6.1 the *scheduling* algorithm's performance stagnates earlier than the performance of the *item-similarity* approach. Therefore, the improvement decreases before settling at around 100% where the number of correctly classified movies also stagnates for the *item-similarity* approach.

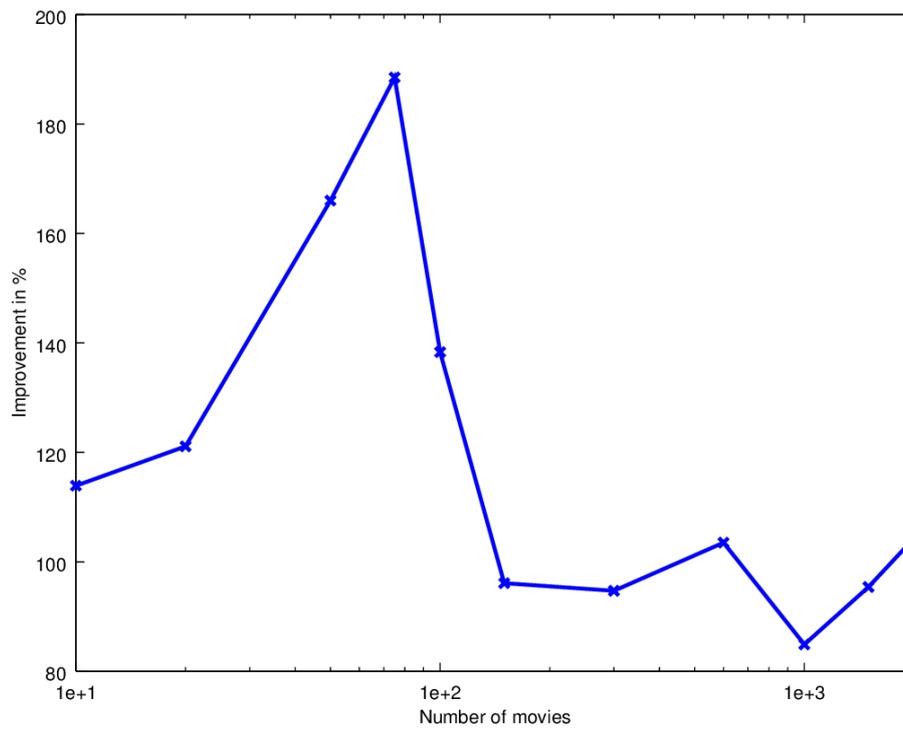


Figure 6.3: The improvement in percent of the scheduling algorithm compared to the item-similarity algorithm.

7

Limitations and Future Work

This chapter contains the limitations which occur on the technical side and also the ones that appeared during the implementation of the scheduling algorithm. Furthermore, the issues that have arisen during the implementation will be explained.

A discussion of future work and possible improvements regarding the scheduling approach and the evaluation will be shown.

7.1 Limitations

The limitations section will be split into the technical limitations regarding the server and the user interface of PEOPLEVIEWS, the limitations of the scheduling approach, and the limitations of the evaluation.

7.1.1 Technical Limitations

The known technical limitations of the system are given due to the used client-server concept and the used *JavaScript APIs* used for the user interface. The following limitations are known

- As many technologies and libraries are used for the user interface, some browsers or versions of browsers might have visualization problems. There are many known issues with any version of the Microsoft Internet Explorer as this browser does not support many of the used APIs, for example, HTML5.
- As the server (see Chapter 4) of the system was written in Java, the server needs a Java installation to be able to run the system.

7.1.2 Limitations of the Scheduling Algorithm

The scheduling approach needs a rather high amount of data to select suitable users (see Chapter 6). The approach needs to know all the items a user interacted with, the micro-tasks she solved in the last period, furthermore, the keywords of items and recommenders, and other properties (see Section 6.1) that are used by the set of equations (see Section 5.4). Therefore it is not easy to apply this approach to other recommender systems.

7.2 Limitations of the Evaluation

As already mentioned in Section 6.1 it is not possible to take every aspect of the scheduling algorithm into account. The reason for this is the used data set. As there are no public data sets available that cover all the aspects needed to exploit the full potential of the scheduling algorithm, some data had to be simulated.

7.3 Future Work

The keyword extraction of the scheduling algorithm could be further improved by labeling the keywords with part-of-speech tags (POS tagging) [51]. This approach assigns one or more specific classes to a word and with this information, it is easier to distinguish between important and not important words. For example, nouns and adjectives could be weighted higher than verbs as they are more descriptive regarding items. POS tagging would reduce the amount of user keywords per user significantly.

Another task for future work would be to use a library for genetic algorithms to tune the weighting parameters of the scheduling algorithm. This would have been beyond the scope of this thesis, therefore, it is subject to future work. This genetic algorithm needs training to tune the parameters and a test set to verify the results. It performs several runs on the data set and would change the weighting parameters of the scheduling algorithm in every run until the number of wrongly distributed micro-tasks is as low as possible. This parameter set with the lowest error is then used as weighting parameters in the algorithm.

A possible problem of learning the parameters is overfitting. As overfitting is also possible for large amounts of data online-learning needs to be applied. In this concept, a possible overfitting can be detected by an increasing amount of not answered or wrongly answered micro-tasks.

Evaluation with PeopleViews data

To be able to perform a study regarding the *scheduling* algorithm which uses the data collected by PEOPLEVIEWS, the following requirements need to be fulfilled:

- The users must not be paid to interact with the system as this influences the data quality and the user's behavior [52].
- Users need to interact with the system repeatedly to generate a history of their interactions and to be able to react to newly assigned micro-tasks.
- The users need to solve micro-tasks as this enables the *scheduling* algorithm to predict the *workload* of users.
- The data collection needs to be conducted in a long-term study for various reasons. The first reason is to give the users enough time to contribute to the system. Second, time-related parts of the *scheduling* algorithm (*importance* and *workload*) can only be evaluated if users repeatedly use the system over a longer period of time.

The requirements regarding the resulting dataset for an evaluation with PEOPLE-VIEWS were already described in Section 6.1.

8

Conclusion

The constraint-based recommender system implemented in this thesis can be used to build and maintain recommenders, add and modify items, evaluate items and generate recommendations. The user can specify supports regarding objective attributes of the items (user attribute values) when evaluating them. These supports are aggregated and used to calculate recommendations as shown in Chapter 3. Furthermore, the user is able to enter requirements to the system to get suitable recommendations.

A completely new user interface was designed with the intention of being intuitive and easy to understand (see Chapter 4). As the system is based on a client-server architecture, it is easy to exchange or improve parts of the user interface.

A game is implemented to motivate users to add knowledge to the system. Furthermore, micro-tasks are used to acquire data to improve the recommendation quality. To assign micro-tasks to suitable users, a set of formulas (see Chapter 5) was introduced that is capable of choosing users based on their previous interactions with the system. These formulas take certain aspects such as a user's interest in a recommender domain, a user's expertise or her current workload into account.

The evaluation of the scheduling algorithm was challenging, as there exist no data sets that met the requirements specified in Section 6.1. The *MovieLens* dataset chosen for evaluation the *scheduling* approach was enriched with data from the *Netflix Roulette* database. Additionally some data was simulated to perform the evaluation as shown in Section 6.1.

The evaluation results show that without tuning the parameters the *scheduling*

algorithm already achieves more than 50% of correctly classified movies. This is a satisfying result for such algorithms. This thesis was a proof of concept that a recommender system can be implemented in such a generic way that it is possible to add multiple recommender domains to one system and distribute micro-tasks for fast and simple data acquisition to users who are suitable to add reasonable data to the system.

Bibliography

- [1] P. Resnick and H. R. Varian, “Recommender systems,” *Commun. ACM*, vol. 40, no. 3, pp. 56–58, Mar. 1997.
- [2] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, “Using collaborative filtering to weave an information tapestry,” *Commun. ACM*, vol. 35, no. 12, pp. 61–70, Dec. 1992.
- [3] A. Felfernig and R. Burke, “Constraint-based recommender systems: Technologies and research issues,” in *Proceedings of the 10th International Conference on Electronic Commerce*, ser. ICEC ’08. New York, NY, USA: ACM, 2008, pp. 3:1–3:10.
- [4] J. S. Breese, D. Heckerman, and C. Kadie, “Empirical analysis of predictive algorithms for collaborative filtering,” in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI’98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 43–52.
- [5] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, “Grouplens: Applying collaborative filtering to usenet news,” *Commun. ACM*, vol. 40, no. 3, pp. 77–87, Mar. 1997.
- [6] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong, “Addressing cold-start problem in recommendation systems,” in *Proceedings of the 2nd international conference on Ubiquitous information management and communication*. ACM, 2008, pp. 208–211.
- [7] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in *Proceedings of the 10th International Conference on World Wide Web*, ser. WWW ’01. New York, NY, USA: ACM, 2001, pp. 285–295.
- [8] G. Linden, B. Smith, and J. York, “Amazon.com recommendations: item-to-item collaborative filtering,” *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, Jan 2003.
- [9] M. J. Pazzani and D. Billsus, *Content-Based Recommendation Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 325–341.

- [10] J. Ramos, “Using tf-idf to determine word relevance in document queries,” in *Proceedings of the first instructional conference on machine learning*, 2003.
- [11] R. Van Meteren and M. Van Someren, “Using content-based filtering for recommendation,” in *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*, 2000, pp. 47–56.
- [12] R. Burke, “Knowledge-based recommender systems,” *Encyclopedia of library and information science*, vol. 69, no. Supplement 32, p. 180, 2000.
- [13] A. Felfernig, B. Gula, G. Leitner, M. Maier, R. Melcher, and E. Teppan, *Persuasion in Knowledge-Based Recommendation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 71–82.
- [14] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara, “The distributed constraint satisfaction problem: Formalization and algorithms,” *IEEE Transactions on knowledge and data engineering*, vol. 10, no. 5, pp. 673–685, 1998.
- [15] A. Felfernig, M. Jeran, M. Stettinger, T. Absenger, T. Gruber, S. Haas, E. Kirchengast, M. Schwarz, L. Skofitsch, and T. Ulz, “Human computation based acquisition of financial service advisory practices,” in *Proceedings of the 1st International Workshop on Personalization & Recommender Systems in Financial Services, Graz, Austria*, 2015, pp. 27–34.
- [16] A. Felfernig, S. Haas, G. Ninaus, M. Schwarz, T. Ulz, M. Stettinger, K. Isak, M. Jeran, and S. Reiterer, “Recturk: Constraint-based recommendation based on human computation,” in *RecSys 2014 CrowdRec Workshop*, 2014, pp. 1–6.
- [17] A. Felfernig, T. Ulz, S. Haas, M. Schwarz, S. Reiterer, and M. Stettinger, “Peopleviews: Human computation for constraint-based recommendation,” in *ACM RecSys 2015 CrowdRec Workshop*, 2015.
- [18] R. Burke, “Hybrid recommender systems: Survey and experiments,” *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370, 2002.
- [19] —, *Hybrid Web Recommender Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 377–408.
- [20] G. Karypis, “Evaluation of item-based top-n recommendation algorithms,” in *Proceedings of the tenth international conference on Information and knowledge management*. ACM, 2001, pp. 247–254.
- [21] A. Albadvi and M. Shahbazi, “A hybrid recommendation technique based on product category attributes,” *Expert Systems with Applications*, vol. 36, no. 9, pp. 11 480 – 11 488, 2009.

-
- [22] G. Lekakos and P. Caravelas, “A hybrid approach for movie recommendation,” *Multimedia Tools and Applications*, vol. 36, no. 1, pp. 55–70, 2008.
- [23] L. von Ahn, “Human computation,” in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, July 2009, pp. 418–419.
- [24] A. J. Quinn and B. B. Bederson, “Human computation: A survey and taxonomy of a growing field,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 1403–1412.
- [25] L. Von Ahn, “Human computation,” Ph.D. dissertation, Pittsburgh, PA, USA, 2005.
- [26] M. C. Yuen, L. J. Chen, and I. King, “A survey of human computation systems,” in *Computational Science and Engineering, 2009. CSE '09. International Conference on*, vol. 4, Aug 2009, pp. 723–728.
- [27] C. Sarasua, E. Simperl, and N. F. Noy, “Crowdmap: Crowdsourcing ontology alignment with microtasks,” in *International Semantic Web Conference*. Springer, 2012, pp. 525–541.
- [28] S. Schnitzer, C. Rensing, S. Schmidt, K. Borchert, M. Hirth, and P. Tran-Gia, “Demands on task recommendation in crowdsourcing platforms—the worker’s perspective,” in *ACM RecSys 2015 CrowdRec Workshop, Vienna*, 2015.
- [29] V. Ambati, S. Vogel, and J. G. Carbonell, “Towards task recommendation in micro-task markets.” in *Human computation*, 2011, pp. 1–4.
- [30] M. Hadano, M. Nakatsuji, H. Toda, and Y. Koike, “Assigning tasks to workers by referring to their schedules in mobile crowdsourcing,” in *Third AAAI Conference on Human Computation and Crowdsourcing*, 2015.
- [31] V. Rajan, S. Bhattacharya, L. E. Celis, D. Chander, K. Dasgupta, and S. Karanam, “Crowdcontrol: An online learning approach for optimal task scheduling in a dynamic crowd platform,” in *Proceedings of ICML Workshop: Machine Learning Meets Crowdsourcing*, 2013.
- [32] X. Wang, “A genetic algorithm for task scheduling based on user overall satisfaction,” in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2012 International Conference on*, Oct 2012, pp. 527–530.
- [33] S. Deterding, M. Sicart, L. Nacke, K. O’Hara, and D. Dixon, “Gamification. using game-design elements in non-gaming contexts,” in *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '11. New York, NY, USA: ACM, 2011, pp. 2425–2428.

- [34] G. Zichermann and C. Cunningham, *Gamification by design: Implementing game mechanics in web and mobile apps.* ” O’Reilly Media, Inc.”, 2011.
- [35] J. Hamari, J. Koivisto, and H. Sarsa, “Does gamification work? – a literature review of empirical studies on gamification,” in *2014 47th Hawaii International Conference on System Sciences*, Jan 2014, pp. 3025–3034.
- [36] D. McSherry, “Similarity and compromise,” in *Case-Based Reasoning Research and Development: 5th International Conference on Case-Based Reasoning, IC-CBR 2003 Trondheim.* Springer Berlin Heidelberg, 2003, pp. 291–305.
- [37] L. Davis, “Handbook of genetic algorithms,” 1991.
- [38] L. Richardson and S. Ruby, *RESTful web services.* ” O’Reilly Media, Inc.”, 2008.
- [39] Apache spark, spark.apache.org.
- [40] C. Bauer and G. King, “Hibernate in action,” 2005.
- [41] R. Johnson, J. Hoeller, K. Donald, C. Sampaleanu, R. Harrop, T. Risberg, A. Arendsen, D. Davison, D. Kopylenko, M. Pollack *et al.*, “The spring framework–reference documentation,” *Interface*, vol. 21, 2004.
- [42] J. L. Carlson, *Redis in Action.* Greenwich, CT, USA: Manning Publications Co., 2013.
- [43] M. Otto and J. Thornton, “Bootstrap,” *Twitter Bootstrap*, 2013.
- [44] B. Bibeault and Y. Kats, *jQuery in Action.* Dreamtech Press, 2008.
- [45] F. Peng, N. Ahmed, X. Li, and Y. Lu, “Context sensitive stemming for web search,” in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval.* ACM, 2007, pp. 639–646.
- [46] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988.
- [47] D. M. Hawkins, “The problem of overfitting,” *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.
- [48] Movielens 10m dataset, grouplens.org/datasets/movielens/10m/.
- [49] Netflix roulette api, netflixroulette.net/api/.
- [50] B. Smyth, *Case-Based Recommendation.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 342–376.

- [51] E. Brill, “Part-of-speech tagging,” *Handbook of natural language processing*, pp. 203–414, 2000.
- [52] S. Chang, F. Harper, L. He, and L. Terveen, “Crowdlens: Experimenting with crowd-powered recommendation and explanation,” 2016.