Development of a Conversational Recommender System in the Context of a Business Travel Chat Bot

Entwicklung eines interaktiven Empfehlungssystems im Kontext eines Chatbots für Geschäftsreisende

Bachelor-Thesis von Nils Dycke aus Gütersloh

Tag der Einreichung:

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Eneldo Loza Mencía

TECHNISCHE UNIVERSITÄT DARMSTADT

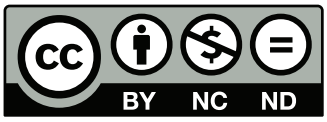
Fachbereich 20 - Informatik
Knowledge Engineering

Development of a Conversational Recommender System in the Context of a Business Travel Chat Bot
Entwicklung eines interaktiven Empfehlungssystems im Kontext eines Chatbots für Geschäftsreisende

Vorgelegte Bachelor-Thesis von Nils Dycke aus Gütersloh

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Eneldo Loza Mencía

Tag der Einreichung:



Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 14. Dezember 2017

(Nils Dycke)

I want to express special thanks to Adrian Weinert, my supervisor at PASS IT Consulting GmbH, who supported me in the acquisition of the data underlying this work and helped me a lot to get access to all necessary technologies.

Abstract

With the rise of electronic flight booking platforms offering an overwhelmingly large and complex set of flight options to their customers, the necessity for an efficient selection and discovery mechanism has become apparent. Especially for the domain of business flight traveling the decrease of time spent searching for an appropriate journey has to be minimized, as this time cannot be used to generate any business value.

In this work we focus on the implementation and evaluation of a recommender system based on machine learning to realize the mentioned selection mechanism through personalization - that is, by offering every customer individually selected items based on their preferences. Additionally, the designed system allows the user to elicit his short-term preferences during a conversational discovery process of the candidate set. This allows the seamless future integration into a chat bot application.

In this document we give an overview on the domain of business flight traveling and offer insights into possible solution approaches for designing a recommender system in this context. From this space of solutions we have selected and implemented a recommendation approach based on probabilistic learning of long-term preferences in combination with an interactive dialog system. The user profiles constructed by the probabilistic component are exploited to determine the entry point for the interactive dialog, which itself consists of a sequence of item propositions in response to the the user's feedbacks - so-called *critiquing*. We prove that this approach is very suitable for the given application requirements and that the implementation performs significantly better than alternative algorithms and baselines. During the subsequent discussion of the recommendation performance of our implementation we give impulses for future research directions and the potential limits of personalization in this field.

Contents

1 Introduction

In this work we will give a brief overview on the technical background of data mining and machine learning techniques. Based upon that we will elaborate on the background of recommender systems described in such a manner that it may serve as a design guideline for a given problem domain. For this purpose we discuss their design spaces focusing on the subset of conversational recommender systems. Relying on these explanations we will analyze the application domain of business flight traveling for the implemented recommender system of this work. Afterwards we will discuss a fitting architecture and implementation for it. Lastly, it will be proven that the developed system is more valuable compared to several baselines and draw a conclusion for further extensions of both theoretical concepts and the developed software.

In this chapter we will briefly motivate this work, introduce related works and describe its goals. Finally, we will give an overview on the application context of the existing software application into which the developed recommender system will be integrated.

1.1 Motivation

Flight booking is a very time intensive task requiring domain expertise, while imposing high cognitive effort on the users. This is induced both by the complex description of the flight journeys and the number of available options. For casual users the exploration of the journey options is either exhaustive or unsatisfactory, as they have to review all options to be certain of the optimality of the booked journey with respect to their needs. For users with a higher level of experience, that is for example frequent fliers, the decision process is less cognitively exhaustive, but always requires the timely expensive exploration of options. This is caused by the dynamics of the underlying flight traveling market. Here regular (i.e. non-personalized) information retrieval mechanisms, searching for a best match from a data mining point of view, are not sufficient to pursue the users' goals on electronic flight booking platforms. They either require to much effort of the user by specifying concrete values or simply lead to unnecessary long searches.
In the context of business flight travelers the aspect of timely efficiency becomes even more important, as time spent on searching for flight journeys reduces the amount of actual work done. Hereby the generated business value is decreased. This narrower domain is especially interesting as it includes users of many different levels of experience which all need support during electronic booking to speed up their decision process.

Here so-called recommender systems, adapting to user preferences for each single booking session and across multiple of those, are highly fitted to meet the described problem of information overload, which is intrinsically connected to this domain. The advances in machine learning and recommender systems in recent years make it possible to employ efficient, adaptive algorithms conquering domains with complex features. The domain of business flight traveling is especially interesting and adequate for researching on the boundaries of employing recommender systems for personalization tasks, as it has both a heterogeneous user base with diverse user preferences and a complex item feature space. Developing a valid recommendation approach for this domain is insightful for various similar domains like train connections or trip planning.

Nevertheless, personalization in the domain of business flight traveling or even the broader domain of flights has not been studied sufficiently or in a manner so that a recommender system could be employed inside an actual business context. Here this work takes place by using a conversational recommendation approach highly suitable for chat-like user interfaces. The implemented recommender system relies on a short- and long-term preference model of the users employing contextualization through a hybrid architecture. As we will prove later on, during the evaluation, a significantly increased performance compared to various baseline systems using standard information retrieval techniques could be achieved. Additionally, a solid theoretical basis for the further development of recommender systems in similar domains was defined.

1.2 Goals of this Work

The goals of this work are to execute extensive research on the design space of recommender systems in general, as well as to develop a suitable model and implementation of a recommender system for the given domain of business flight traveling. The resulting recommender system as a software application will be integrated into an actually employed software system hosted by the company PASS IT Consulting residing in Aschaffenburg, Germany.

In a first step a clear definition and analysis of the personalization problem and its formalization lie at the core of the development of a guideline for modeling and implementing the recommender system. This analysis includes a detailed model of recommender systems and the parameters of the application domain influencing the design space of valid systems. In front of the background of the development of the recommender system various filtering techniques are analyzed and discussed with respect to their strengths and weaknesses. By a comprehensive review of so-called conversational recommendation and its implementation we go into more details for this investigation.
In a second step the developed theoretical background serves as a basis for modeling the users and items of the business flight traveling domain. This also forms the starting point of the analysis of the adequacy of the developed model and possible implementing algorithms. The resulting architecture is defined in a general manner, so that it can be transferred to similar domains.

Finally a solution system is implemented as a promising prototype and compared to various baselines during the evaluation. This comparison involves the evaluation of each system component in isolation and in combination. Hereby the adequacy of the overall approach is determined and serves as a basis for the conclusion of its applicability for business flight traveling and similar domains.

1.3 Related Work

In the previous sections we gave an overview on the goals and motivation of this work. For now there has not been developed a recommender system for the domain of business flight traveling. For the broader domain of flight traveling (i.e. including both holiday and business trips) there have been several propositions in the literature which, however, focus on different recommendation goals and techniques. In this section we will briefly introduce these systems and compare each to the approach of this work.

One related proposition found in the literature is the so-called *Smart Client* [42] employing an iterative process for elicitation of the user's needs. After specifying an initial query, the options are presented to the user by various graphical figures and views on the candidate space. In response the user may add an explicit constraint on one feature to reduce the number of candidates. This work of Pu et al. differs in the basic recommendation approach requiring the user to define explicit values as constraints. This, we argue, requires higher domain profession compared to the approach of this work and increases the cognitive effort imposed on the user. Additionally, no across session user adaption takes place.

A well-known paper describes the design of the so-called *Adaptive Trip Adviser* [26], which is a conversational recommender system embedded inside a chat-like interface using natural language. The user receives an initial recommendation based on the preferences expressed in past sessions and may refine the proposed item until it meets his preferences using constraints and relaxations. The mentioned approach differs from this work by two central aspects: Firstly, a profile built across sessions is used for deriving an implicit user query, instead of exploiting the profile as a form of refinement of a recommendation based on an explicit query. Secondly, the approach for the iterative preference elicitation dialog, which was taken by the authors, is significantly different allowing other forms of feedback.

Finally, there is one proposition of a graduation project [6] describing the recommendation of potentially interesting destinations for travelers. In the background an information retrieval engine determines a ranked list of flights sorted by their degree of similarity to a given user query. Here the feature space is defined based upon domain knowledge from various sources and data mining results on the set of available flights. Afterwards a list of destinations from the top resulting flights is recommended to the user as an alternative. Hereby the recommendation goal of the described work differs essentially from the one of this work. Additionally, contrary to this work, an approach not considering user preferences across sessions or the iterative elicitation of user needs was taken.

All in all, there exist no recommender systems for the domain of business flight traveling, yet various implementations for the broader domain of flight traveling were proposed. These, however, take a significantly different approach compared to the one of this work. In addition, many of them are evaluated transparently resulting in a low comparability or reproducibility of their results.

1.4 Cooperation with PASS IT Consulting

PASS IT Consulting GmbH is a German company residing in Aschaffenburg, which offers business travel solutions to other companies. For this work a cooperation was conducted to design a recommender system which can be actually employed in the business context in the future to further improve the developed system and the underlying theoretical concepts. As the research on recommender systems is intrinsically application oriented such a cooperation is a central asset to scientific work in this field. In this section we will give a short overview on the surrounding software application hosted by PASS IT Consulting and the future intended use to contextualize the developed recommender system.

The current surrounding software application is an electronic flight booking platform offered to the employees of contracted companies. The customers specify a query and receive a list of flight journeys in response. This list can be ordered according to typical criteria like price or the point in time of the journey. Underlying this application an information retrieval engine hosted by an external third party provides a list of flight journey options in response to the query. The concrete functionality of this engine is unknown as it is kept secret for its business value. The implemented recommender system is integrated as a form of middleware refining the results of the retrieval engine by personalization mechanisms. The surrounding application and the integration of the developed software are described with more detail in subsection 3.1.2.
The future intended integration of the developed recommender system takes place within a chat bot system, which is an automated assistant interacting with the user in natural language. The concrete interface design and underlying intelligence are the topic of future work.

Additionally, a central asset to this work was that the evaluation of the developed recommender system was realized on real user data to ensure its validity. The executed evaluation bases on anonymized booking data collected during the hosting of the mentioned system. This data could only be acquired thanks to the fruitful cooperation.

2 Recommender Systems

In this chapter we will first discuss data mining and machine learning techniques, which are necessary to comprehend the following explanations on the basics of recommender systems. In this second step we will first introduce recommender systems with a goal-oriented analysis of their possible configurations, afterwards discuss the so-called filtering techniques employed within recommendation algorithms and finally have a closer look on problems and extensions in such systems.

2.1 Relevant Data Mining Techniques

In this section we will give a brief overview on the subset of relevant data mining and machine learning techniques referred to for the implementation of the recommender system of this work. We will first introduce similarity measures and the concept of association mining along with its well-known implementing algorithm. Finally, we will describe a subset of Naïve Bayes classifiers used in the implemented recommender system.

2.1.1 Similarity Measures

In data mining and especially the field of case-based reasoning (CBR) *similarity measures* play a central role, as they define, which items and problem instances are similar to each other, respectively. We will now give a very brief overview on the definition and notion of similarity measures.

Formal Definition of Similarity Measures

We call a function meeting the following definition a similarity measure [3]. This differs from the more restrictive definition of similarity metrics included by the more general set of similarity measures [27].

Definition 2.1 (Similarity Measure) *Let $\mathscr{A} = A_1 \times ... \times A_n$ be the space of features A_1 to A_n rendering all possible descriptions of items of a given domain. Further on let $R = [0,1] \subset \mathbb{R}$ be the range of real values from 0 to 1.*

A function sim specified by $sim : \mathscr{A} \times \mathscr{A} \to R$ is called a similarity measure. Then you say for $i, j \in \mathscr{A}$ items of this feature space that $sim(i, j)$ defines the (degree of) similarity for the items i and j.

Usually we associate a distance measure with every similarity measure as the multiplicative inverse [10]. The following definition of distance measures specifies this relation.

Definition 2.2 (Distance Measure) *Let $\mathscr{A} = A_1 \times ... \times A_n$ the space of features A_1 to A_n rendering all possible descriptions of items of a given domain. Further on let $R = [0,1] \subset \mathbb{R}$ be the range of real values from 0 to 1. Let $sim : \mathscr{A} \times \mathscr{A} \to R$ be a similarity measure.*

Then the function dist is called the associated distance measure for sim and it is define by the following equation. You can specify both sim and dist completely by defining only one of them.

$$dist(a,b) = \frac{1}{sim(a,b)} - 1 \iff sim(a,b) = \frac{1}{1 + dist(a,b)}$$

Selection of Similarity Measures

As Althoff et al. [3] point out for most application domains there exist various possible and reasonable similarity measures. Yet most similarity measures are not useful for involving them for any given task. For example, a similarity measure comparing pictures of iris with each other will most likely perform bad for comparing faces. This is why the quality of similarity measures can only be defined relative with respect to their appropriateness for a task.
For non-numerical domains *weighted hamming measures* are typical similarity measures. Such a measure is given by the following definition [3]. As they can be adjusted and composed individually they are a useful and flexible tool during the development of a valuable measure for a given task.

Definition 2.3 (Weighted Hamming Similarity Measure) *Let $\mathscr{A} = A_1 \times ... \times A_n$ be the space of features A_1 to A_n rendering all possible descriptions of items. Further on let $R = [0,1] \subset \mathbb{R}$ be the range of real values from 0 to 1. Finally may $a_i \in A_i$ denote the i-th component of an item $a = (a_1, ..., a_n) \in \mathscr{A}$.*

The similarity measures $sim_i : A_i \times A_i \to R$ for $i \in \{1, ..., n\}$ are called the local similarity measures. Then all similarity measures $ham : \mathscr{A} \times \mathscr{A} \to R$ on the whole feature space specified by the following equation are called weighted hamming measures.

$$ham(a,b) = \sum_{i=1}^{n} \alpha_i \cdot sim_i(a_i, b_i) \text{ for the weights } \alpha_1, ..., \alpha_n$$

Additionally, for specifying the local similarity measures we will refer to the well-known euclidean similarity measure (e.g. defined in [17]) during the discussion of the implemented conversational recommender system. The one-dimensional euclidean similarity is applicable for numerical domains. This underlying euclidean distance measure is a very intuitive measure for determining the distance between two points in some space.

Definition 2.4 (Euclidean Similarity) *Let a, b be numerical values. We then define the euclidean similarity measure euclidsim according to definition 2.2 over the euclidean distance.*

$$euclidsim(a,b) := \frac{1}{1 + \sqrt{a^2 + b^2}}$$

2.1.2 Association Rule Mining

Rules are used in classification and data mining to specify relationships between items or item features. Here we will give a brief overview on the subclass of association rules and the Aproiori algorithm for discovering such rules in sets of structured item features.

Definition of Association Rules

As association rule mining is historically connected to market basket analysis we usually refer to the elements of a given data set as *transactions*. While every transaction consists of multiple *items*, which can be interpreted as categorical features of each transaction. Then association rules are defined as follows [55].

Definition 2.5 (Association Rules) *Let $\mathscr{S} \subseteq \mathscr{P}(S)$ be the dataset of transactions as a set of subsets of items. Then a rule $A \to C$, where the antecedent A and the consequent C are arbitrary sets of values from the space of items $\mathscr{P}(S)$, is called an association rule. The intended semantics of an association rule is that transactions containing the items in A always contain the items in C, as well.*

Using association rules structural relationships can be specified amongst items, which can be understood as the features of transactions. There exist two most common measures for describing the quality of association rules: The first is the *support value* and the second is the *confidence value* of an association rules for a given dataset. Intuitively speaking, the support of a rule defines its relevance in the data set, while the confidence specifies its certainty. Formally they are defined as follows [55].

Definition 2.6 (Support and Confidence) *May $\mathscr{S}$ be the set of transactions and $a := A \to C$ is an association rule. May $\mathscr{S}_{AC} \subseteq \mathscr{S}$ denote the set of transactions for which the feature values A and C are all given. Lastly let $\mathscr{S}_A$ denote the set of transactions for which the feature values of A are given. We then define support and definition as*

$$support(a) = support(A,C) := \frac{|\mathscr{S}_{AC}|}{|\mathscr{S}|} = \frac{\#\text{ transactions with A and C}}{\#\text{ transactions in total}}$$

$$confidence(a) = confidence(A,C) := \frac{|\mathscr{S}_{AC}|}{|\mathscr{S}_A|} = \frac{\#\text{ transactions with A and C}}{\#\text{ transactions with A}}$$

From a probabilistic point of view the support value of a rule defines how likely it can be applied; that means what proportion of transactions fulfills the rule in total. And the confidence value specifies the likelihood for a transaction having the items A to have the items C, as well. These measure will become relevant for selecting a subset of association rules from all possible association rules on a transaction dataset in the following algorithm.

The Apriori Algorithm

The Apriori algorithm is an efficient algorithm for generating a set of association rules on a given set of transactions that meet the thresholds for minimal support and confidence. The Apriori algorithm can be described as a two step process [55], which we will describe very briefly:
Firstly, all item subsets of the transactions in a database are considered and discarded if they have too low support. This process, however, is executed in a more sophisticated manner than pure enumeration: Starting at the smallest item sets containing one item the sets containing one more element than in the previous iteration are considered. Only those sets are carried on for the next iteration that are supersets of the sets of the previous iteration meeting the minimal support requirements.
Afterwards from each of the items sets association rules are generated by iteratively shifting all combinations of subsets from the consequent to the antecedent. For each combination the confidence is calculated and only association rules above the minimal confidence boundary are added to the result.

Extended implementations of the Apriori algorithm exist which are supposed to find a fixed number of rules with maximum support values. As in its basic implementation maximization of the support is not per se included multiple rounds with an decreasing support threshold are executed [55].

2.1.3 Naïve Bayes Classifier

Naïve Bayes classification is one of the most basic, probabilistic classification approaches, which is nevertheless very often employed in machine learning applications. We will give a brief overview on its theoretical background, classification concept and properties.

Theoretical Background of Naïve Bayes Classifiers

The class of Naïve Bayes classifiers is motivated by and based on the Bayes' rule from the field of decision theory and stochastic. In its most general form it is defined as follows [55].

Definition 2.7 (Bayes' Rule) *Let A and B be events and may $P(x)$ denote the probability for observing event x, while $P(x|y)$ denotes the conditional probability of observing x given y. Further on assume that $P(B) > 0$, i.e. the event B is observable. Then the Bayes' rule states that the following equation holds:*

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Modeling the classes of items in a supervised learning scenario as observable events, as well as understanding the component values of feature vectors representing items as events lead to the following formulation of the Bayes' rule for classification. In classification we are interested in determining the probability of a given sample with given features to belong to a certain class, which is expressed in the conditional probability of the following equation's left hand side.

Definition 2.8 (Bayes' Rule for Classification with Feature Independence Assumption) *Let $C = \{C_1, ..., C_n\}$ be the set of observable classes in a supervised machine learning scenario. Further on may $F = F_1 \times ... \times F_m$ denote the observable feature space of items. We will understand each vector component's value as an isolated event and let $P(x, y)$ denote the probability of the events x and y to be observed together. Finally we also assume the feature values' probability distributions to be independent. Then the Bayes' rule for classification for a class $c \in C$ and a feature vector $(f_1, ..., f_m) \in F$ is defined as:*

$$P(c|f_1, ..., f_m) = \frac{P(f_1, ..., f_m|c) \cdot P(c)}{P(f_1, ..., f_m)} = \frac{P(f_1|c) \cdot ... \cdot P(f_m|c) \cdot P(c)}{P(f_1) \cdot ... \cdot P(f_m)}$$

Changing the perspective to an empirical probability point-of-view we can estimate the probability of an item described by the features $(f_1, ..., f_m)$ to belong to class c by calculating the probabilities of the right hand side from training data with class labels. This calculation is done (in the simple case) by counting all samples with the given features or classes and dividing it by the number of all samples.

This rule directly induces a classification scheme by selecting the most likely class given a feature vector. The Naïve Bayes classifier is hence defined upon the following decision rule.

Definition 2.9 (Naïve Bayes Classifier Decision Rule) *Let $C = \{C_1, ..., C_n\}$ be the set of observable classes in a supervised machine learning scenario. May $(f_1, ..., f_n)$ be an observed feature vector of a given item. Then the Naïve Bayes classifier selects the class with the highest probability according to the Bayes' rule for classification with feature independence assumption:*

$$\arg \max_{c=C_1,...,C_n} P(f_1|c) \cdot ... \cdot P(f_m|c) \cdot P(c)$$

Extensions of Naïve Bayes Classifiers

There exist several extensions and derivations of the basic Naïve Bayes classifier increasing the performance or robustness of the classifier.
Smoothing is an approach to prevent the case that the Naïve Bayes classifier inappropriately estimates the probability of a class to be zero only because one single feature value has never appeared together with this class label in the training data [17]. This can be inappropriate, if the other feature probabilities are in favor of this class, but the single feature's conditional probability in the denominator of formula 2.8 is zero overwhelming all other probabilities. The simple trick of smoothing is to add 0.1 to each feature and class count during the determination of the conditional probabilities. Hereby a feature's probability for a class, which was not observed in the training set, is nevertheless above zero, but does not bias estimator towards any class.

For many classification tasks the class membership of items of the training data can be imbalanced. This means that for at least one class only few items of the dataset are associated with it and hence the classifiers performance with respect to predicting this, so-called *minority class*, is very low. To tackle this problem the concept of *over-* and *under-sampling* was developed [15]: In classical over-sampling mechanisms the training instances of the minority class are duplicated by a certain scheme. In the trivial case a random subset of samples is chosen and added again to the training data without altering them. In classical under-sampling the majority class, i.e. the class with more present samples, is reduced by discarding certain items of this class. Again, this can be done in a sophisticated manner or trivially by random choice.
A special form of minority over-sampling is the SMOTE approach [15] (synthetic minority oversampling technique) where a randomly chosen subset of items from the minority class is used to generate synthetic items. These artificial instances are derived such that they lie on the classification boundary towards a majority class. Concretely, for each selected minority item the closest item of the majority class according to a similarity metric is determined. Then a synthetic item with feature values randomly chosen from the values between those of the two items is generated and labeled as the minority class.

2.2 Recommender Systems

The term *recommender system* (RS) describes a multi-faceted concept, which is treated in the literature with subtly different notions. Due to the fact that the topic of recommender systems unites many fields of research, there exist many different perspectives on its definition and even more answers on the question, "How to develop an adequate recommender system for a given domain?". In this section we are treating those two complex questions and give detailed insight into the field of research on recommender systems.

For this purpose we will first introduce the general concept of information filtering systems, define recommender systems informally and formally and afterwards have a closer look at the algorithms used for the generation of recommendations. Finally we will define and discuss the subset of so-called conversational recommender systems, as this is of high importance for our implementation, and give an overview on measures to evaluate the quality of RS. All steps will, in the end, serve the purpose of defining a guideline for developing the recommender system of this work.

2.2.1 Information Filtering & Personalization

To gain a deeper understanding of the context of recommender systems and their definition it is important to have clear knowledge on the concept of personalization and how it is transferred into computer scientific terms.
We will first introduce information filtering systems and the information overload problem as key concepts to contextualize the idea of personalized systems.

Information Filtering

Information retrieval (IR) and *information filtering* (IF) are closely related fields of research, both addressing the problem of retrieving a subset of most relevant items with respect to a user's needs from a large item catalog [44]. Here items a considered as distinct objects and the catalog of items a collection of those objects. However, as Belkin et al. [8] (as cited in [44]) explain information filtering systems differ from IR systems in the fact that the users' degree of domain profession is assumed to be low and that their needs are figured as non-well-defined. This means that an information filtering system is intrinsically designed as a user-centered selection mechanism, adapting to the complex, dynamic and individual user needs [20]. In this work we will consider the term of information filtering by this definition. We usually refer to the described property or behavior of IF systems as *personalization*. We will discuss this term in the next paragraphs in more detail.

Information Overload Problem

In the context of IF systems the term *information overload* or *information overload problem* borrowed from the field of psychology was introduced to describe a decision making scenario where an individual or group of individuals is overwhelmed by a huge set of options and their accompanying information [20]. In such scenarios, like it can be found in e-commerce or the World Wide Web in general, information filtering systems are necessary to support the user in his personal decision making process and overcome the information overload problem.

User Preferences & Tastes

In the previous paragraphs we already used the term user need, which is referred to as *user preferences* or *user taste* when talking about information filtering systems, often, however, without giving a precise definition. All of the above terms refer to the basic assumption that a human user of an information filtering or retrieval system lacks certain information he intends to acquire through the usage of the system [20]. Additionally it is commonly assumed, as supported by results of the field of psychology, that the user's preferences are persistent over a period of time [4] and can be (partially) implied by his interaction with the system [57]. From now on, the terms user preferences, needs and tastes will be used interchangeably and with this semantics.

Personalization

For the above definition of personalization, as a user individual selection mechanism, a clear description of what is *individual* is missing. To resolve the vagueness of this term we will now contrast and illustrate personalized and non-personalized selection mechanisms as they are usually understood to in the literature:

Personalized Selection Following Janach et al. [31] we consider a selection mechanism to be personalized, if each user receives a different set of selected items depending on his taste. However, it should be irrelevant how the users' tastes are acquired; meaning that both manually provided needs by the user, so-called *user customization* [40], and automatically deduced needs from past user behavior, so-called *rule-based* selection [40], can be the basis for personalized selection mechanisms. For example, an online bookstore offering to input favorite authors to select books most probably liked by the current user is personalized. While, of course, a selection mechanism deducing this information from the current user's purchase history is personalized, as well.

Non-Personalized Selection Selection mechanisms which rely on the mapping of each user onto the same subset of items, like for example by offering always the most frequently chosen ones of all users, are considered non-personalized by us. This is consistent with the basic literature (ref. [31], [20]). For example, an online bookstore requiring the user to specify a feature value for a book he desires (a title, an author etc.) to show him the best match for the given input is rather a non-personalized system. This differs from the above example, as the user input is not interpreted as an manifestation of his preferences, but simply as a filter criterion. Another obvious example would be an e-commerce website suggesting the same, most bought item to all users.

Having this distinction in mind, we will now be able to introduce recommender systems in an adequate, formal manner. Additionally, this definition will serve as a classification scheme of the underlying problem in the domain of business flight traveling.

2.2.2 Definitions of Recommender Systems

After we have independently introduced information filtering systems we will now give an overview on three exemplary definitions of recommender systems in the literature and end with a definition we will use in this work. This will serve as the basis for the following sections defining models of components and parameters of RS (in 2.2.5), as well as give a precise definition of the problem solved in recommender systems.

Recommender System Definition by de Gemmis et al. [20]

De Gemmis et al. give the most basic definition of recommender systems by associating them directly as IF systems, extending the definition we gave in subsection 2.2.1 and elaborating typical components of RS:

> "*Information Filtering Systems, like Recommender Systems, [...], adapt their behavior to individual users by learning their tastes during the interaction, in order to construct a profile that can be later exploited to select relevant items.*"

On one hand this definition is very useful as we can simply transfer all statements on information filtering systems directly to recommender systems. On the other hand, this definition is very abstract and reduces a RS to user profiles and an exploitation algorithm for generating recommendations. Their context of use, purpose and concrete application is not included.

Recommender System Definition by Ricci et al. [48]

The definition of RS by Ricci et al. has a different, more practical view on recommender systems:

> *"Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user [...]. A RS normally focuses on a specific type of item (e.g., CDs, or news) and accordingly its design, its graphical user interface, and the core recommendation technique used to generate the recommendations [...]. RSs are primarily directed towards individuals who lack sufficient personal experience or competence to evaluate the potentially overwhelming number of alternative items that a Web site, for example, may offer."*

This definition emphasizes the application aspect and user-centered design of recommender systems. While it becomes apparent that RSs are mostly designed for specific item domains and included into broader applications, the formal aspects of recommender systems and their recommendation algorithms are neglected.

Recommender System Definition by Adomavicius et al. [1]

In contrast to the very abstract and the practical definition of recommender systems Adomavicius et al. define RSs by introducing the formal *recommendation problem*. We will have a closer look at the formal definition of this problem in the following subsection. Intuitively the authors interpret a recommendation for a user as the evaluation of the mapping from the set of items and the set of users to a totally ordered set of scores. By that definition recommender systems approximate this underlying mapping based on a subset of known input-output-pairs.

This definition, of course, does not consider the many complex aspects of a whole recommender system and focuses on a formal model of recommendation algorithms and the input and output data types.

A Unifying Recommender System Definition

As all of the above definitions each have a very different perspective on recommender systems we give a unifying definition containing all mentioned aspects. For the rest of this document we will refer to RS by this definition:

Recommender systems (RS) are information filtering systems. They are defined for a specific, but dynamic set of items and users of which every element can be specified by certain, domain-dependent features. A RS consists at least of a recommendation algorithm, a profile for each user and a model for each item. These components in combination solve the recommendation or prediction problem to recommend the most valuable item to a current user.
RS are commonly embedded inside a software application offering an item catalog to users and can be closely connected to the user interface of this system.

2.2.3 Recommendation and Prediction Problem

As mentioned in the previous subsection we can model the problem solved by recommender systems or rather their recommendation algorithms formally. Using this model we can later introduce measures for the quality of RS, discuss recommendation algorithms and possible extensions in a clear manner and with consistent designators throughout this document.

Symbols and Designators for the Formal Model

Before we discuss the different problems solved by recommendation algorithms we introduce formal symbols and designators which we will use in this document.

Definition 2.10 (User Space) *We call the set of distinct users the user base or user space and refer to it by $\mathcal{U}$. We assume this set to be finite.*

Definition 2.11 (Item Space) *We call the set of distinct items the item space or item catalog and refer to it by $\mathcal{I}$. We assume this set to be finite.*

Prediction, Recommendation and Ranking Problem

According to Janach et al. [31] two types of problems in the context of recommending items can be defined, which we will explain and formalize in the following: The first is the *recommendation problem* as defined by Adomavicius et al. [1] and the second is the *prediction problem* closely related to the first problem. For completeness, we argue, one should add the *ranking problem* often referred to as *top-n-recommendation* [9].

Intuitively speaking, generating a recommendation (i.e. solving the recommendation problem) is the selection of the most relevant item to the user. Solving the prediction problem, on the other hand, is the calculation of a score for each

item in the catalog. Usually this score is referred to as the *utility* of the item and is used as a basis to determine the most relevant item for generating recommendations. As an extension to the recommendation problem the ranking problem is defined as the determination of a list of items in order of decreasing relevance to the user, where the first item is the most relevant one.

Differing between the mentioned concepts is more precise and necessary for defining further aspects of recommender systems. It is noteworthy that there are many authors using all of those terms interchangeably or at least without a clearly cut definition. For example in [7] and [39] recommendation is only the binary selection problem by relevance, while in [2] there is a differentiation between prediction and a generated recommendation. Alternatively sometimes recommendation is treated as what we defined as ranking, like for example in [30].
We will now model the discussed, intuitive descriptions formally by the following definitions leaned onto the central paper by Adomavicius et al. [1].

Definition 2.12 (Prediction Problem) *Let R be a totally ordered set. Then let $u : \mathscr{U} \times \mathscr{I} \to R$ be an unknown utility function assigning each user-item-pair $(c, i) \in \mathscr{U} \times \mathscr{I}$ a score $r \in R$ representing the actual degree of relevance of the item i to user c.*

The prediction problem is defined as the approximation of the utility function u from a finite set $N \subset (\mathscr{U} \times \mathscr{I}) \times R$ of defined input-output-pairs onto the whole set of user-item-pairs. The set N of input-output-pairs for the utility function models the known ratings for items by users.

Definition 2.13 (Recommendation Problem) *Let $u : \mathscr{U} \times \mathscr{I} \to R$ be a utility function approximated by solving the prediction problem.*

The recommendation problem is defined by the following equation:

$$\forall c \in \mathscr{U} : i_c = rec(c), \text{ where } rec : \mathscr{U} \to \mathscr{I} : c \mapsto \operatorname*{argmax}_{i \in \mathscr{I}} u(c, i)$$

Hereby the recommendation problem is given by the prediction problem and the maximization problem on the item space of the utility function for each user of the user space.

Definition 2.14 (Ranking Problem) *Let $n \in \mathbb{N}$ be the length of the ranking list and $u : \mathscr{U} \times \mathscr{I} \to R$ be a utility function approximated by solving the prediction problem.*

The ranking problem is defined by the following equation:

$$\forall c \in \mathscr{U} : i_c = rank(c),$$

$$\text{where } rank : \mathscr{U} \to \mathscr{I}^n : c \mapsto (l_1, ..., l_n),$$

$$\text{where } l_1 = rec(c) \text{ and for } k \in \mathbb{N} \text{ with } 1 < k \leq n : u(c, l_k) = \max_{i \in \mathscr{I} \setminus \{l_1, ..., l_{k-1}\}} u(c, i)$$

Hereby the ranking problem is given by the prediction problem and the maximization problem for each list member on a reduced, considered item set of the utility function for each user of the user space.

Recommendation Algorithm

As the formal model of the recommendation problem obviously abstracts from many important aspects of a recommender system, like for example the mechanisms to collect the ratings for user-item-pairs we will refer to these problems only in formal contexts. For this purpose we distinguish the term *recommendation algorithm* referring to the algorithm at the core of a RS solving the recommendation problem from the recommender system as a whole. Compliant with many authors we will understand the recommendation algorithm in a broader sense and use this term for algorithms solving the prediction and ranking problem, as well. In section 2.3 the different approaches for implementing recommendation algorithms will be discussed based on the above definitions.

2.2.4 Model of Parameters of Recommender Systems

Recommender systems consist of many complex, interacting components and adjustable parameters, which have to be chosen and composed wisely during the development process to design a useful system for the given problem domain [41]. In this subsection we will discuss models for recommender system describing it through parameters later using them as a basis for a design guideline of RSs. For this purpose we will have a look at two parameter models for RS found in the literature and discuss their adequacy.

Data-oriented Model by Janach et al. [31]

As argued by Janach et al. a recommender system significantly changes in its implementation with a change in the underlying input data. Therefore RSs, as a whole, should be modeled in three layers: The domain-dependent input parameters, the recommendation component, at the core of the recommender system, and the recommendation list as the output of the recommendation process. The recommendation component is defined as a black-box-function mapping from the input parameters onto a list of items together with their scores - the recommendation list. We will now give a overview on these input parameters in a condensed form.

User Profile and Contextual Parameters The user profile or user model refers to the mapping of the user's tastes onto a structured representation usually based on his feedback for consumed items. The contextual parameters extend this model by ad-hoc information on the situation in which the user is interacting with the system.

Community Data The community data is derived from the set of all user profiles. This data may be used in combination with the current user's profile to generate a recommendation.

Product Features The product features are a representation of the properties of the items. The data per item can be stored in an either structured or unstructured way and may be derived from the complex contents of the item or directly resemble the item's features.

Knowledge Models Janach et al. understand the term knowledge model as a formal representation (e.g. rules) of domain expertise. This knowledge may be acquired through data analysis or from domain experts.

The authors emphasize that not all of the input parameters are relevant for all kinds of recommender systems, which we will later introduce as filtering techniques, yet they can always form part of the recommendation process. Additionally some of those parameters can be freely chosen by the designer of the system, but most of them are strongly determined by the application domain.

It is apparent that this model is primarily data-oriented as it focuses on the input data available to the recommendation algorithm. Yet the process of acquiring and mapping the information of the application domain inside the given, framing application of the RS are not discussed; for example the mechanisms to collect feedback on items to build a user profile are not considered. Hence this model has to be extended by a more algorithm- and application-oriented view to receive a valuable basis for a design guideline. Such an extensional model is the one described with more detail in the following paragraph.

Application-oriented Model by Picault et al. [41]

In their paper Picault et al. give a general guideline describing steps in the design process of a recommender system with respect to an application domain. The underlying model of RSs is therefore mostly user- and application-oriented and split into fields of design decision, which in turn can be straightforwardly transferred into parameters. In the following we will give a compact, slightly reduced overview on those parameters based on the guideline of Picault et al.
Recommender systems are strongly connected to their application environment and depend on its parameters. The environmental parameters of a RS can be grouped into three dimensions: The user model, the data model and the application model. In this context the user model refers to the behavioral model of the consumer of the recommendation service, the data model to the domain model of the items and the application model to the assumptions on the integration of the RS into the surrounding software.

The user model is divided into nine detailed factors which we summarize by this four parameters: User characteristics, goals, expectations and context. In table 2.1 those parameters are compactly described. They basically model the domain-specific properties of how the users behave and what is known about them.

Parameter	Compact Description
User Characteristics	Demographic description of the users and user groupings
User Goals	Motivations and tasks pursued by users
User Expectation	Degree of the users' goal-orientation and expectation in the accuracy of the RS
User Context	Possible access devices, social and situational usage conditions

Table 2.1.: Table of the parameters of the dimension of user model together with a brief description. The contents are derived from the RS design guideline [41].

The data model is made up of ten detailed factors again summarized through the following four parameters: Available item features, available metadata on the items, properties of the overall item set and feedback acquisition mechanisms. These parameters are briefly described in table 2.2. The parameters of the data model dimension define, in their very essence, how the items of the domain can be reasonably described and how user feedback may be included.

Parameter	Compact Description
Available Item Features	Information and data types given on each item from all data sources
Available Metadata on Items	Information on complex items deducible from their raw data together with this metadata's amount and value for characterizing the item
Properties of the Item Set	Number, distribution of features among the items and dynamics of the item set
Feedback Acquisition	Availability and nature of feedback on items by users

Table 2.2.: Table of the parameters of the dimension of data model together with a brief description. The contents are derived from the RS design guideline [41].

The application model consists of eight factors, which can be summarized by two relevant parameters: The content navigation interface and the performance criteria. These parameters basically define, what possibilities exist to involve feedback ad-hoc into the recommendation process, as well as how the recommendation is presented to the user and in which performance dimensions the RS can be described and optimized.

Parameter	Compact Description
Content Navigation Interface	Options and context of presenting recommendations and adapting it to user interaction
Performance Criteria	Criteria for measuring the quality of the RS from all stake-holders perspectives

Table 2.3.: Table of the parameters of the dimension of application model together with a brief description. The contents are derived from the RS design guideline [41].

Obviously this model of parameters of recommender systems focuses on the application domain emphasizing the mapping from the informal domain properties onto formal data structures and algorithms. In general this model seems adequate, as it treats all aspects of recommender systems and their environment with a useful degree of detail. For the next subsection this parameters will render our basis for deducing a model of components of RS from it.

2.2.5 Model of Components of Recommender Systems

As proposed in the previous subsection 2.2.4 we can determine a comprehensive and useful model for parameters influencing the design and quality of recommender systems. Those parameters refer in the first place to the application domain's properties influencing the possible design space of recommender systems and only secondarily to the actual design of the components and data structures of the RSs. Therefore we will deduce a component-oriented model of recommender systems in the following, which will later enable us to discuss aspects of RS in a structured manner. In this subsection we will consider the application-oriented model by Picault et al. [41] and refer to it by this name.

Transferring the User Model Dimension

The user model parameters (see table 2.1) can be split into those which describe non-functional requirements on the RS and those which represent a description of input data. The user expectations belong to the first category, while user context, characteristics and goals to the second one. For our component-oriented model we will ignore non-functional requirements descriptions as they cannot be mapped onto functional or data units. We derive the following data components of RS from the remaining three parameters:

User Model The user model or profile is the formal representation of a user and his preference information.

User Model Construction Algorithm The user model construction algorithm is the algorithm mapping raw input data as given by the domain onto the user model of the RS. We understand this term in the broader sense of the functional module triggering and executing profile updates.

User Space Model The user space model is the formal characterization of quantity and quality of the user base. This can include application knowledge on the users or simply be a compact representation of the set of all user models.

Context Model The context model formally specifies the context of the user including his currently pursued task or goal.

Context Construction Algorithm The context acquisition algorithm is the algorithm determining the current user's context from raw data as given by the application domain. Again we understand this in the broader sense of the functional module triggering and executing the context profile construction.

Context Space Model The context space model formally specifies the set of possible contexts and their relation to user goals, preferences and in last consequence the recommendation. This can be a compact representation of the set of all context models or a more sophisticated model built upon domain knowledge.

Transferring the Data Model Dimension

The data model parameters (see 2.2) are grouped into those referring to the items themselves and the one referring to the acquisition of feedback. We can directly transfer them to the following components:

Item Model The item model is the formal representation of all relevant information on each item of the domain.

Item Model Construction Algorithm The item model construction algorithm is the algorithm mapping raw input data on items as given by the domain onto the item model of the RS.

Item Space Model The item space model represents knowledge on items and the distribution and relation of features amongst the items, as well as the compact description of the set of all item models.

Feedback Model The feedback model formally specifies the feedback which can be given by a user on one item.

Feedback Model Construction Algorithm The feedback model construction algorithm is the algorithm constructing feedback models for items from the raw data given by the user. We understand this in the broader sense of the functional module triggering and executing the update of the feedback profiles.

As the application model dimension only consists of non-functional components, we will do not consider it here.

Recommendation Components

To complete the model of components, of course, the recommendation algorithm at the core of the RS has to be included. As the result of the recommendation can be either a ranked lists, single and multiple items or item-score-pairs (as given by the formal problems 2.2.3) we also consider the output data type of the algorithm as a single component.

Recommendation Model The formal representation of the result of the recommendation, which can be either a ranked list, one score per item or only a subset of items.

Recommendation Algorithm The recommendation algorithm as specified in sub-section 2.2.3 mapping the items on the recommendation model for each user.

The resulting component-oriented model is depicted in figure 2.1. The real users, contexts, items and transactions (that is user interactions with the RS) are represented as ovals. By the respective construction algorithms they are mapped onto formal data structures which are the input to the recommendation algorithm. The user, item and context space models serve as a parameter for the recommendation. The designed model can be viewed as a more detailed and process-oriented extension to the models found in [31] and [20].

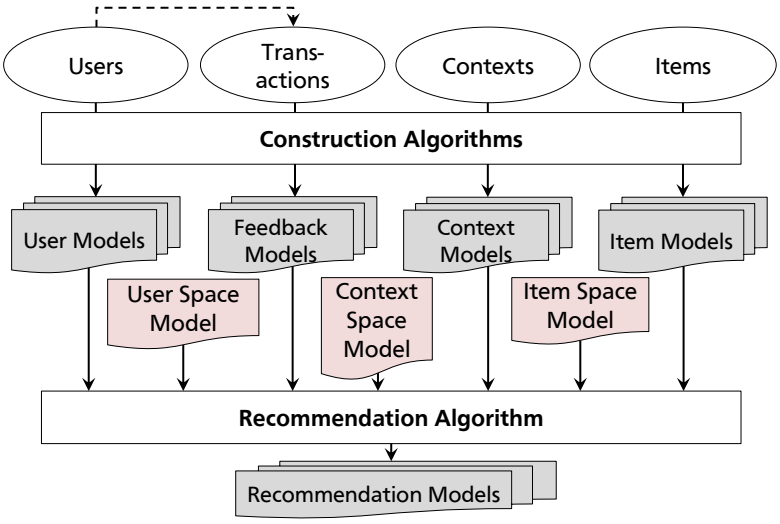


Figure 2.1.: Schematic diagram of the component model for recommender systems. The algorithms are depicted by white boxes, real entities as circles and data structures in gray or red. The arrows indicate the processing flow and relation between the components.

2.2.6 Feedback Mechanisms in Recommender Systems

In the previous two subsections we discussed parameters influencing the design of recommender systems and introduced a model of components of RSs. One key aspect of designing the components of the RS is the choice of the feedback model and the mechanisms for acquiring feedback from the user. This is the case, as it imposes cognitive effort on the user of a information filtering system, which he might not be able or willing to invest [20].

In this subsection we will give a brief overview on the term feedback. Afterwards we define the kinds of feedback and their acquisition common in recommender systems together with their up- and downsides for the quality of the RS.

Notion of Feedback in RS

We follow Ricci et al. [48] and understand feedback as all forms of user-controlled interactions with the recommender system collected over time. This can be either an explicitly given rating for an item or maybe just a sequence of accesses to item descriptions. As a side note, this rather general definition, including feedback which is more than a simple rating for an item, is called *transactions* [48]. In that context it is noteworthy, as well, that feedback is commonly used interchangeable with the term item-rating as some score representing the degree of relevance of an item is the most natural way to represent the users opinion on an item.

For this reason we will introduce three types of mechanisms for acquiring and transforming user input into item-ratings in the following. For a structured description we will take the perspective of the model of components of recommender systems (see subsection 2.2.5). Therefore we will differentiate between the users' raw input and the feedback model as the result of the construction algorithm on the input data.

Types of Feedback Mechanisms

There are three general types of mechanisms for gathering a score for an item by the current user: *Explicit*, *implicit* and *hybrid* feedback mechanisms [31]. The first two types differ in what raw user input is collected and as a consequence in the feedback construction algorithm. The third type includes all mechanisms combining the first two types. We will no give an overview on these mechanisms and discuss their strengths and weaknesses.

Explicit feedback mechanisms generate a score for an item by directly asking the user for a rating [48]. The given raw input hereby is equal to the feedback model and the construction algorithm is trivial. The most obvious downside of this approach is that the user experience can be reduced due to the higher effort during the usage of the RS. Usually sparsity or incorrectness of user ratings is the result. On the other hand, the confidence of the user into the RS can be increased, as he can directly relate his ratings to the recommendations [30]. Usually the explicit asking for ratings is figured to result in precise feedbacks because this is a direct translation of the user's expressed needs [31]. However, this is questionable considering that the user of an IF system is assumed to have a lower degree of domain knowledge and a less clearly determined goal. These are requirements for giving reliable, explicit ratings. Here especially the choice of the rating scale can have a great influence on the quality of feedback [31].

Implicit feedback mechanisms generate a score for an item by inferring it from the current user's interaction with the RS or rather the whole surrounding application [48]. There exists a plethora of approaches to realize this goal [38]. For example, all queries during a shopping session of a user on an e-commerce platform can be logged and afterwards all items with a description containing those keywords may be rated as relevant. An alternative and common way of inferring scores is by observing the purchases or consumptions of the user determining those items as relevant. The clear strength of this approach is that it does not imply more effort from the user, as his interaction with the system is influenced in no matter [30]. As a result the sparsity of ratings tends to be no problem [31]. On the other hand, a reliable model of the user behavior is required to interpret his actions reliably rendering this mechanisms generally less accurate [31]. For example, buying an item on an e-commerce platform is an ambiguous indicator for the relevance of this item, because it could have been bought as a present for a different person. At the same time, it can be argued that implicit ratings reveal the real user's preferences with a high accuracy as for the explicit feedback mechanisms the collection method itself influences the rating of the user [31].

Hybrid feedback mechanisms employ both explicit and implicit feedback mechanisms combining them to compensate each others weaknesses [30]. For example, implicit feedbacks are collected to prevent the sparsity problem, while also asking for item ratings. The combination, however, of the ratings of the different approaches is not trivial, as for example different scales can be used and normalization is generally necessary.

Feedback Rating Scales

By the definition of the prediction problem in subsection 2.2.3 it is the task of a recommendation algorithm to approximate the utility function mapping from the pairs of users and items to a totally ordered domain R. As all of the above feedback mechanisms result in the rating of an item for the feedback model they can be directly interpreted as the output for a given user and item of the utility function. Because R is not further specified, there are many options to define total ordered rating scales. Here the most common ones described by Schafer et al. [51] (as cited by [48]) are listed:

Numerical Scale A scale as a subset of the real numbers is given. Usual are discrete, natural scales, like for example 5- or 7-point scales. Alternatively in some systems continuous and negative scales are used [31].

Oridnal Rating Ordinal ratings are very similar to numerical scales on the real numbers except for the fact that meaningful identifiers for the degree of relevance are introduced. Examples for such identifiers are "highly relevant", "relevant", "irrelevant", etc.

Binary Rating Binary ratings are a degraded form of ordinal ratings with only the two values "relevant" and "irrelevant" or "like" and "dislike".

Unary Rating This type of rating can be viewed as a degraded form of ordinal ratings with the two values "relevant" and "no information". This differs from binary ratings in the fact that only positive ratings indicating the relevance for an item can be given, while missing ratings (or "no information") indicate only that there is no knowledge on the preferences for this item.

The usage of the different scales in recommender systems can depend on the application domain and the recommendation algorithm. There exist domains where explicit feedback is not available rendering unary and binary feedback through item consumption logging the natural way of feedback acquisition. For example for the data underlying the evaluation of this work described in section 4 only unary ratings are available making it necessary to extrapolate them to receive binary rating classes. On the other hand, for explicit feedback mechanisms ordinal scales can offer a better orientation for users than the quite abstract numerical scales.

2.2.7 Challenges in Recommender Systems

In the following paragraphs we will give a brief overview on several issues and challenges for the design of recommender systems as we accumulated them from the literature. This list enables us to discuss up- and downsides of different recommendation algorithms and the quality measures for RSs in the following sections.
For the sake of clarity we will structure the issues and challenges by their reference to either the feedback, user and item space or the preference learning process.

Feedback-related Issues and Challenges

Issues and challenges with regard to missing or wrong feedback exist in multiple variants rendering any learning, personalized recommendation process difficult:

Sparsity Problem Adomavicius et al. [1] describe the problem of sparsity or rating sparsity as follows: "In any recommender system, the number of ratings already obtained is usually very small compared to the number of ratings that need to be predicted". This means that overcoming the lack of feedback for items by the current user and by other users (only for recommendation approaches relying on other users' interests) is a universal problem for RSs.

New User Problem The new user problem is often referred to as one facet of the *cold start problem* and describes a special case of the sparsity problem. Here a new user is added to the system and therefore initially does not have any ratings on items given making useful recommendation difficult [29].

New Item Problem The new item or early rater problem is referred to as the second facet of the cold start problem. It describes a special case of the sparsity problem, where a new item is added to the catalog and initially no user has given ratings on this item. For certain recommendation approaches this may lead to the situation where this item is never recommended to any user [20].

User- and Item-space-related Issues and Challenges

Issues and challenges with regard to the user or item space evolve around, statistically speaking, unusual users or items, as well as the limited processing capabilities of the item and user space:

Gray Sheep Problem The gray sheep or unusual user problem is introduced by de Gemmis et al. [20] as the phenomenon that for user's with uncommon tastes the generated recommendations tend to be inappropriate as they are optimized for the majority of users having the same, popular tastes. This is basically an overspecialization phenomenon of recommendation algorithms. The name is motivated by the observation that the user base can be split into "black sheep" (unusual users) and "white sheep" (usual users), while the recommendation algorithm treats them as the same resulting from blending their properties - hence treating all users as "gray sheep".

Limited Content-analysis For many item domains no structured data describing the items' features are given; for example for textual or multi-media domains. Therefore automatic or manual content specification is necessary, which can be either very costly or results in poor item descriptions. Additionally, very different items may be mapped onto the same set of features again leading to poor recommendations [1].

Limited Item Space Coverage In many applications it can be considered as valuable to recommend items from the whole item space, yet it is not unusual that recommendation algorithms tend to consider only a small subset of most popular items as a phenomenon of overspecialization [48]. Avoiding this restriction is a challenge for the design of recommender systems.

Scalability Problem The scalability problem arises for certain recommendation algorithms which increase massively in their computation times, if the user or item space grows [20]. For todays fields of application, like in e-commerce, often algorithms dealing with huge item and user bases are necessary. For example, the recommender system of amazon.com had to potentially deal with about 67 million products of the biggest category of items in 2016 [19] and with 300 million users in 2015 [18].

Preference-learning-related Issues and Challenges

As many recommender systems are based on adaptive algorithms learning user preferences by additional ratings, challenges for the adequate design of the learning process over time arise:

Overspecialization & Serendipity Problem The problem of overspecialization refers to the phenomenon that for some recommendation approaches "the user is limited to being recommended items that are similar to those already rated" [1]. This can render the recommender system as significantly less useful to the user, if he is interested in exploring new options. The term of serendipity was introduced to name the contrary property of RSs to propose unusual items different from previous ones. Hence the overspecialization problem is also called the serendipity problem [20].

Flexibility of Profile Adaption Anand et al. [4] give a conclusive introduction to the necessity of modeling the user through short- and long-term preferences. They explain that a recommender should on one hand reflect the human short-term memory according to today's psychological knowledge and offer a highly flexible, adaptive component for recommendation, as well as one that can perform recommendations based on long-term preferences acquired over many sessions. This also might involve the discarding of old, obsolete feedback.

The above description of challenges and issues related to the design of recommender systems can clearly be applied to any recommender system. Yet there exist many problems relevant to only a subset of special recommendation techniques. In the following section we will see which of the so-called filtering techniques tend to produce which of the issues. Together with the previously introduced component model a basic guideline for designing and assessing RS is already given.

2.2.8 Integration and Application of Recommender Systems

In the previous subsections we primarily focused on the theoretic aspects of recommender systems. Because RSs are intrinsically connected to surrounding software applications and due to their practical-oriented nature it is important to have insight into the integration of RS into existing software.

In this subsection we will first give a very brief overview on the recommender systems from a user's point of view including the possible incorporation of a RS into existing software applications from a user interaction perspective.

User-centered Analysis of the Integration of Recommender Systems

If we understand recommender systems as black boxes returning ranked lists of relevant items to the current user then many questions arise on their integration into whole software applications. As IF systems are by definition user-centric we will focus on two aspects of the integration of RS with a strong influence on the user experience. These two aspects are motivated by the following two questions oriented towards the presentation of recommendations to the user:
When to set-off the recommendation process and present its result to the user? How to present the recommendations to the user and how to enable the navigation through complex result sets? In the following paragraphs we will discuss possible design options answering these questions.

Proactive and Reactive Recommendation

With respect to the question of when to set-off the recommendation process Bridge et al. [11] differ between two types of recommender systems: *Reactive* and *proactive* RS. Reactive recommender systems only generate a result as a response to an explicit user request usually formulated as a *query* describing the desired product. On the other hand, proactive RS do not accept any user-input for a parameterized computation, but may be triggered at any time and offer a set of most relevant items purely according to the formal recommendation problem. Proactive recommender systems may be used to expand the interests of the user and inspire him to consider items he has never perceived as relevant, while reactive ones are rather a supportive mechanism improving the decision, finding and search process of the user.

Single-shot and Conversational Recommender System

The second question on the presentation of recommendations becomes relevant, if the result sets are big or the presented items are characterized by complex features. Either property might lead to the overwhelming of the user despite the application of an information filtering mechanism. Here we differ between RS which only generate a single set of potentially relevant items without any interaction or refinement mechanisms. These type are so-called *single-shot* recommender systems and they have the potential to leave the user overwhelmed or unsatisfied, if the single recommendation is not adequate. Opposing this concept lie the *conversational recommenders*, which generate multiple recommendations during an interactive dialog to refine the result until the user is satisfied by the proposition [11]. While this is a good option for domains with complex items this imposes cognitive overhead on the user actively reflecting over his needs. We will further investigate the design space of those systems in section 2.4.

From the above introduced categories for RS from a user interaction perspective it becomes apparent that there are multiple options for integrating a RS into an existing application. However, not all of them are useful for the concrete item domains and the design has to be chosen dependent on the application domain.

2.2.9 Extensions to Recommender Systems

After the detailed analysis of basic recommender systems, their design space and application of the previous subsections we will now give a brief overview on what we call *extensions* to RSs. This term is borrowed from the name of the groundbreaking paper of Adomavicius et al. [1] proposing extensions of the classical, that is basic, implementations of recommender systems. Other authors frequently refer to this as *recent developments* or *emerging topics*. We figure the term extensions, however, as more neutral over time. We will focus on contextual recommendation, as it is the most relevant form for our work, because it lies at the bottom of the implemented RS.

For this purpose we will introduce contextual recommendations as a form of an extension discussed in the literature. First we will have a closer look at the basic concept of contextual recommendations, afterwards formalize this concept and finally overview the different paradigms found in contextual recommendation.

Concept of Contextual Recommendation

The core idea of *contextual recommendations* or *context-aware recommender systems* is to incorporate the current users contextual information into the recommendation generation. The notion of context in this definition is not clearly cut and their exists a plethora of interpretations from different fields of research [2]. Anand et al. [4] give one possible, computation-oriented definition of context:

> "*This definition [of context] includes the state of the user, state of the physical environment, state of the computational environment and history of user-computer-environment interaction.*"

As an illustration, the location of a person is often a relevant part of his *current state* from, for example, the perspective of a restaurant recommender system, while for some systems the noise level around him can be considered a relevant description of his *physical environment*. The state of the *computational environment* can be, for example, the device he uses to access the recommendation service, while the history of interactions may be quantified in clicks per minute. All of those properties can be used to determine the current goals and intension of the user in the given situation, but are not themselves the input, but rather parameters to the generation of recommendations. Refer to [22] for a generalization of possible features for characterizing the context in the above dimensions.

It is important to note that in the above illustration two subtle assumptions were made: The first is that context is a distinguishable, measurable property stable over a certain period of time during the use of the RS. The second is that it can be modeled with respect to some specified task by structured attributes [4]. This view on context is called the *representational view* by Dourish et al. [21] (as cited by [1]), while a contrary position, the *interactional view* states that context is not directly observable, but only inferable from the user behavior without a fixed set of attributes describing it. In the end, both views can be useful for describing different aspects of the user's context and may be combined for an adequate model.

Formalization of Contextual Recommendation

Having the above definition in mind we will now introduce the paradigms for involving context in the recommendation process, while we abstract the manner in which the context is modeled and inferred.
In its very essence, for context-aware RS the underlying prediction problem is extended to the space of context. In the following we will refer to the set of all possible, distinct contexts by *context space* and write it as $\mathscr{C}$. We receive the *contextual prediction problem* by the following definition after [2]:

Definition 2.15 (Contextual Prediction Problem) *Let R be a totally ordered set. Then let $u : \mathscr{U} \times \mathscr{I} \times \mathscr{C} \to R$ be an unknown utility function assigning each user-item-context-triple $(c, i, b) \in \mathscr{U} \times \mathscr{I} \times \mathscr{C}$ a score $r \in R$ representing the actual degree of relevance of the item i to user c in the current context b.*

We define the contextual prediction problem as the approximation of the utility function u from a finite set $N \subset (\mathscr{U} \times \mathscr{I} \times \mathscr{C}) \times R$ of defined input-output-pairs onto the whole set of user-item-context-triples. The set N of input-output-pairs of the utility function models known ratings for items by users in certain contexts.

The contextual recommendation and ranking problem are analogously deduced from the contextual prediction problem as described in subsection 2.2.3. This formalization will serve as a basis for specifying the RS of this work in a precise manner (see subsection 3.2.2).

Paradigms of Contextual Recommendation

This formal definition, however, does not describe, how classical recommendation algorithms can be extended to involve context. We differ between three paradigms for the incorporation of context into the recommendation process [2]. We will now briefly introduce them. For the sake of clarity we will refer to the underlying data of input-output-pairs by N as in the definition of the contextual prediction problem and to recommendation algorithms and input data of the ordinary prediction problem without context by the adjective *regular*.

Contextual Pre-filtering Here the data set N is pre-processed in a two-step process: First all input-output-pairs, where the context component is different from the current context of the user, are discarded. Afterwards this set containing only elements with the same context is reduced to input-output-pairs, by ignoring the third component, suitable for the regular utility function. Finally the regular recommendation algorithm is applied.

Contextual Post-filtering For this paradigm the data set N is first reduced to a regular set of input-output-pairs ignoring context. This, in turn, is used as the input for a regular recommendation algorithm generating a list of recommendations or a ranked list of items. This result is used as an input for a filtering algorithm based on the current context and the data set N. This algorithm selects only those items which are the most valuable for the current user in the current context according to information deduced from N.

Contextual Modeling Here the feature space of the items is extended by the contextual attributes. For every considered item during the recommendation the current context's features are transformed, included and are later stored together with a ranking in the data set N upon which the recommendation algorithm is defined. In this sense, the recommendation algorithm is still a regular one, as it maps from the user and the extended item features onto a score. The algorithm, however, usually has to be adapted to treat the new input features correctly.

All of the above paradigms have their strengths and weaknesses and it is an important design decision to select one of them. There are many restricting factors for the possible approaches for incorporation of context into a RS, starting at the underlying recommendation algorithm and the definition of context. Having a look into the literature, contextual modeling is quite common (e.g. found in [39]) due to its straight-forward design. Contrary to this, for existing (i.e. already designed and implemented) RS the filtering methods are advantageous, as they can be simply added around a given non-contextual recommendation algorithm. For a complete discussion refer to the paper of Adomavicius et al. [2].

For additional information on extensions in recommender systems the reader may refer to the "Recommender Systems Handbook" [49] giving detailed and comprehensive introductions by experts of their fields into a majority of possible extensions for recommender systems. For example, a complete list of emerging topics and the discussion of group recommendations, social RS, explanations and items sequence recommendation can be found in addition to the contextual extension discussed above. For an overview on security issues and RS in ubiquitous computing environments one might read the introductory book by Janach et al. [31].

2.3 Filtering Techniques in Recommender Systems

In the previous section we gave a detailed introduction of recommender systems and decomposed them into several, elementary components. We will now elaborate several basic approaches on how to implement the recommendation algorithm relying on different user and item models. This analysis will be split into informal and formal definition forming the basis of the subsequent discussion of the adequacy of each approach for a given problem domain.

In the beginning we will give an overview on these approaches in form of a taxonomy for recommender systems. Afterwards, for each approach, we will introduce the underlying idea, give a concrete definition and discuss variants for realizing it. To receive a valuable design guideline we will also give insight into the strengths and weaknesses of each.

2.3.1 Taxonomy of Recommender Systems by Filtering Techniques

Commonly recommender systems as a whole are classified by the so-called filtering or recommendation technique which they employ. This term refers to the underlying concept including the type of input necessary for generating the recommendation, as well as the algorithm itself.

In the following we will first briefly introduce the set of filtering techniques and discuss variations of their definitions found in the literature. To enhance the understanding of the relation of those approaches with respect to their input and underlying concepts we will afterwards discuss a taxonomy for filtering techniques (and hereby for RSs) which is a useful basis structuring the solution space for a practical guideline to design RSs.

Overview on Filtering Techniques

Although there exist many different definitions of the set of filtering techniques in RS all propositions can be viewed as variations of the following list of techniques as specified in [20].

Colllaborative Filtering (CF) Collaborative filtering RS rely on two simple ideas: The first is that each user's tastes are best specified by their history of ratings for items. This especially disregards the domain given item descriptions. The second idea is that users with similar tastes in the past will have similar tastes in the future [31]. Hence CF recommendation algorithms are sometimes referred to as community-based approaches.

Content-based Filtering (CBF) The underlying ideas of content-based filtering RS are that items are best specified by their domain-specific features and that the user will prefer items in the future which are similar to the ones preferred in the past. The item features are derived from the actual, unstructured contents of the items (e.g. the text of a news article or the audio file of a song) and are not domain-given in a structured way.

Knowledge-based Recommendation (KBR) Knowledge-based recommender systems are defined upon the idea that both users and items can be specified through structured feature vectors. It is assumed that these can be matched with respect to their degree of relevance by a utility function relying on functional domain knowledge.

Demographic Recommendation (DR) The idea of demographic recommendation is that firstly users can be grouped by their domain-specific properties into stereotypical classes and secondly those classes can be associated with tastes. Therefore DR RS determine recommendations through grouping the users into demographic classes and recommending each member the preferred items of his assigned class.

Hybrid Recommender Systems (HRS) Hybrid RS rely on the idea of combining multiple of the above filtering techniques by for example aggregating their recommendation results. There exists a multitude of approaches for uniting different recommendation algorithms into one hybrid system.

For this work we will stick to this five filtering techniques, as they represent a superset of all variations of sets of filtering techniques of this field of research. In the literature, though, content-based and knowledge-based RSs are commonly not differentiated (e.g. in [1] or [30]), because both refer to the description of contents of items regardless of whether their domain-specific description is given in an unstructured or structured way [40]. In this manner, it can be argued that CBF should be considered a sub-class of KBR approaches due to the fact that content-based filtering approaches only add a preprocessing step extracting structured specifications of unstructured item descriptions to the knowledge-based recommendation [31]. We perceive this perspective to be more adequate, which is why we will differentiate the five described filtering techniques, but will also consider CBF approaches to be a form of KBR approaches. We will discuss this hierarchic relation in the following paragraph.

Input-oriented Taxonomy of Filtering Techniques

We developed the following taxonomy based on the model of components described in subsection 2.2.5. The following two observations motivated the definition of an *input-oriented* taxonomy of recommender systems according to their employed filtering techniques: Firstly, the selection of the filtering technique lies at the core of the development of a recommender system, because this already strongly determines the strengths and weaknesses of the system. Additionally, the chosen technique limits the space of possible recommendation algorithms. Secondly, as described in subsection 2.2.4, the input parameters given by the domain should have strong influence on the choice of the algorithms of a RS adequate for its field of application. Hence a taxonomy of filtering techniques by their types of input models (user and item models) is a useful orientation for selecting a recommendation approach based on the domain-specific input parameters.

One can generally differ between two types of models in the above described filtering techniques: *Behavioral* or *feedback-based* models and *non-behavioral* or *domain-based* models. While behavioral models describe items or users by their history of ratings or interactions, non-behavioral models rely only on external domain-specific knowledge and describe the user or item statically. Using these terms we can describe every filtering technique in its two dimensions, the user and the item model.

Starting at collaborative filtering it becomes clear that both the item and the user model of CF approaches are of a behavioral nature, because they only consist of a history of ratings per user and every item is only specified by the users' ratings for it. Secondly content-based filtering approaches represent the items by their domain-specific features derived from their contents. Therefore the item model is non-behavioral as no feedback or interaction, but only domain knowledge is considered to build an item profile. However, the user profile is of a behavioral nature because it is aggregated by the user's feedback on items and their features.

For knowledge-based and demographic RS this classification scheme is not as obvious as for CBF and CF. Having a look at knowledge-based recommendation we receive by its very strict definition that both items and users are represented by structured sets of features derived from domain knowledge. If we consider CBF a form of KBR, as described above, we can associate both domain- and feedback-based features with knowledge-based recommendation.
Demographic recommendation relies on domain-specific user profiles describing their demographic properties, rendering it non-behavioral. Whereas the nature of the item model is behavioral, as the items are not described by their domain-specific properties, but rather by a profile of which demographic classes prefer this item. Although there exist many demographic RS, where the item profile is manually and statically specified, this is nevertheless only a feedback-based profile, as the preferences of a group of users for each item is analyzed based on a behavior analysis of this group. This assignment can, however, be criticized depending on the concrete approach. Lastly, hybrid recommender systems can have, of course, user and item models of both types.

From this two-dimensional description of the filtering techniques we receive a useful taxonomy of them, which is depicted in table 2.4. Here the user and item model dimensions are assigned one or more techniques in a tableau scheme. We will refer to this taxonomy for categorizing recommender systems during the discussion of the implementation of the RS of this work by directly associating them with their employed filtering technique.

In the following subsections we will analyze the different filtering techniques with more detail. Our aim is to give a general overview on each approach and to define a set of decision criteria for selecting a filtering technique for a given set of requirements from the application domain.

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Item Model</td></tr>
<tr><td>Domain-based</td><td>Behavioral</td></tr>
<tr><td rowspan="2">User Model</td><td>Domain-based</td><td colspan="2">Knowledge-based Recommendation</td><td>Demographic Recommendation</td></tr>
<tr><td>Behavioral</td><td>Content-based Filtering</td><td>Collaborative Filtering</td></tr>
</table>

Table 2.4.: Table giving an overview on the taxonomy of filtering techniques by their input types. Every filtering technique is assigned a class according to its item and user model types. Hybrid recommender systems are not included as they would belong to every class.

2.3.2 Collaborative Filtering

Collaborative filtering is from a historic perspective one of the most important filtering techniques, as it was one of the first approaches applied to the recommendation problem, which was thus studied intensively over the last decades [31]. We will treat this topic with less detail on concrete implementing algorithm, but rather focus on the general up- and downsides of this approach. This will form a basis for evaluating its appropriateness for the application for the implementation of the RS of this work.

In this subsection we will first give a detailed definition of collaborative filtering and its underlying formal concepts. From this definition we will group the space of CF algorithms in classes with different algorithmic properties and finally discuss the intrinsic strengths and weaknesses of CF recommender systems.

Definition of Collaborative Filtering

It is difficult to give a precise definition for the variety of collaborative filtering algorithms; either the definition is to vague or does not include the important class of item-based CF algorithms, which we will introduce later. We will stick to the definition by Janach et al. [31] which walks on the fine line between being precise and being too narrow:

> "*The main idea of collaborative recommendation approaches is to exploit information about the past behavior or the opinions of an existing user community for predicting which items the current user of the system will most probably like or be interested in.*"

We can extend this definition by a formal specification of collaborative filtering algorithms. For this purpose we will introduce the underlying concept of the user-item-matrix beforehand: All CF algorithms rely on the so-called *user-item-matrix*, which can be simply viewed as table where every user is represented by a row and each item by a column. Hereby every cell of the table represents the score the user of the row gave to the item of the column [51]. This intuitive interpretation is shown in figure 2.2 and is formalized by the following definition.

Definition 2.16 (User-item-matrix) *Let $n \in \mathbb{N}$ be the cardinality of the set of users $\mathscr{U}$ and $m \in \mathbb{N}$ be the cardinality of the set of items $\mathscr{I}$. Hence we can enumerate these finite sets with fixed indexes $\mathscr{U} = \{c_1, ..., c_n\}$ and $\mathscr{I} = \{i_1, ..., i_m\}$.*

Further on, let u be the utility function, where $u : \mathscr{U} \times \mathscr{I} \rightarrow R$ defined as in the recommendation problem of definition 2.13. Then we call the matrix $M \in R^{n \times m}$ the user-item-matrix for the utility function u or the recommendation problem of u, if and only if for every index $k \in 1, ..., n$ and $l \in 1, ..., m$ it holds that

$$M_{k,l} = u(c_k, i_l)$$

Like for the utility function the user-item-matrix is not given completely, meaning that only a subset of entries is known and the rest is called empty. It is noteworthy that an empty entry does not reflect a negative or zero rating, but only a missing rating. We can use the user-item-matrix to formally define collaborative filtering algorithms.

Definition 2.17 (Collaborative Filtering Algorithm) *Let M be the user-item-matrix for the utility function u of a given recommendation problem, then we call an algorithm $cf algo$ a collaborative filtering algorithm, if it solves the given recommendation problem while using only a matrix $M' \in (R \cup \{\epsilon\})^{n \times m}$ as an input, for which it holds: $\forall k \in \{1, ..., n\} : \forall l \in \{1, ..., m\} : M'_{k,l} = M_{k,l} \vee M'_{k,l} = \epsilon$ where ϵ denotes an empty entry in the matrix.*

Having this definition based on the user-item-matrix in mind the two basic classes of collaborative filtering algorithms seem obvious. They differ in the manner in which the underlying matrix is evaluated to estimate the missing ratings. Algorithms estimating missing ratings for items (by the current user) by comparing the current user's row with the rows of other users are called *user-based*, while the approach estimating the ranking for the current user on some unrated item by comparing this item's column with the columns of other items are called *item-based* [51].

We will give a very brief introduction to those two classes and their possible implementing algorithms. We will, however, stay on an abstract level for our explanations as CF recommender systems are not considered in detail for the implementation of the RS of this work.

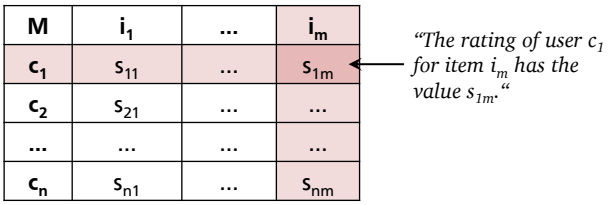


Figure 2.2.: Illustration of the user-item-matrix as a table. Every row represents one user and every column one item. The value of each cell is defined as the ranking of the user of the row for the item of the column of the cell.

User- and Item-based Collaborative Filtering

The basic idea of user-based CF is to understand the user-item-matrix as a set of users characterized by ratings for each item. This set in turn can be used like a case-base to find similar users to the current one. We usually call this group of users the *peer users*. Now the set of best rated items aggregated from all peer user profiles, which are not yet rated by the current user, can be extracted as a recommendation [1].
The most known algorithm realizing this strategy is the *k-Nearest-Neighbors-Algorithm* (kNN-Algorithm) [51] using as the underlying, best performing similarity metric *Pearson's correlation coefficient* [31]. We will not go into the details of this algorithm and refer to the data mining book by Witten et al. [55] for a general discussion of the kNN-Algorithm and to Adomavicius et al. [1] for a discussion of its application in the domain of CF recommender systems.

Underlying item-based CF is the idea of evaluating the user-item-matrix column-wise, that means viewing at it as a set of items specified by the ratings given from all users. Now for all non-rated items for the current user the score is predicted in a two step process: Firstly, the degree of similarity towards each rated item is determined and stored. Then a weighted average is computed by summing the ratings of the rated items, each normalized by their stored degree of similarity to the currently analyzed item. The result is that items similar to the most preferred ones according to their user ratings receive a high score [31].
This strategy is commonly realized via the k-Nearest-Neighbors-Algorithm, as well. [51]. Here the *cosine similarity measure* offers the best results as the underlying similarity measures for determining the set of nearest items [31]. It is

noteworthy that for this approach pre-computed similarity values for all items reduce the complexity of the recommendation process and are adequate over a longer period of time, as in general the opinion of all users on one item does not change rapidly over small periods of time [31].

For the described two paradigms their exists a plethora of implementing algorithms besides the named, best known algorithms. For a more detailed taxonomy of CF algorithms and a discussion of their trade-offs in precision and timely performance one might want to consider the introduction to CF recommender systems by Janach et al. [31] or the comprehensive overview by Schafer et al. [51]. To name the most important terms in this context, one might investigate *model-* and *memory-based* CF algorithms, as well as the concept of *matrix preprocessing* or have a look at *probabilistic algorithms* solving the recommendation problem of collaborative filtering RS through classifiers or regressors from the field of research of data mining and machine learning.

Strengths and Weaknesses of CF Recommender Systems

After the introduction of the algorithmic approaches found in collaborative filtering systems we will now shine a light on the strengths and weaknesses of CF recommender systems, which directly follow from its nature of modeling users and items only by the given feedback. We will start off with the positive aspects of CF RS and afterwards name several downsides of those systems. We will refer to many of the challenges named in subsection 2.2.7.

In the first place, CF RS have the benefit of being domain-independent [31] because they do not rely on the domain-specific features of the items. Thereby heterogeneous item domains with items described through different features or with partially missing attribute specification (problem of limited content-analysis) can be dealt with in CF recommender systems.
Additionally, it is noteworthy that collaborative filtering RS is the filtering technique probably most deeply researched and with the possibility of comparing the algorithms of different item domains [31]. Both of this aspects result in a generally high reliability in granting a solid recommendation performance for any domain. In addition, it is noteworthy that the scalability problem is usually properly tackled through pre-computed similarity measures in item-based CF algorithms [31]. Therefore CF RS can be applied for large item and user sets without a great increase in the computation time for generating recommendations.
Lastly, collaborative filtering recommender systems have the advantage of naturally being able to recommend items significantly different from one another. That means items different from previously liked ones with respect to their *domain given* features; hence offering serendipity and flexibility in the adaption to abrupt preference changes [57]. This is the result of comparing items by their rating vectors completely independent of their actual content or description.

There exists a set of scenarios connected to the introduced challenges, in which CF recommender systems perform generally worse than RS employing different filtering techniques. Collaborative filtering algorithms decrease significantly, if the user-item-matrix is very sparse, because neither user- nor item-based approaches can reliably identify peer groups. The special cases of sparsity, the new user and new item problem, cannot be dealt with, as well [20]. This means that CF algorithms are generally not adequate for domains with highly dynamic item and user bases. In addition to that, for domains where items are consumed rarely, that means there are long time spans in between two ratings, like e.g. buying computers, CF approaches can only offer obsolete recommendations even if the user-item-matrix is not sparse [31]. This is the direct consequence of the assumption that user preferences were absolutely stable over time made in CF, which is not adequate.
Further on, the gray sheep problem is intrinsically connected to CF as the recommended items are determined by the tastes of other users [20]. Here for users with rare tastes in user-based approaches no sufficient peer group can be determined or for item-based approaches the set of similar items is difficult to determine due to sparse ratings by the majority of users.
Finally, one should mention that collaborative filtering recommender systems completely lack transparency of recommendations from a user's perspective [20]; this means, that the user can have decreased trust in the results because he does not know how they are influenced by him. This it the direct consequence of the abstract nature of representing items and users by feedback vectors.

We draw a final conclusion on the reasonable application of CF recommender systems from the above discussion: Collaborative filtering algorithms perform very well on domains with insufficient, domain-specific item descriptions and have the advantage of recommending completely different items. On the other hand, those systems tend to make too general recommendations, often propose inappropriate items based on obsolete feedback and most importantly deal very badly with new items and users.

2.3.3 Content-based Filtering

The recommendation algorithms of content-based filtering are historically connected to text classification as an instance of the general concept of extracting a semantic specification of the contents of an item described by unstructured data [31]. As discussed in subsection 2.3.1 on the taxonomy of RS and filtering techniques, we will understand them as a subset of knowledge-based recommender systems. Using the words of Towle et al. [53] content-based filtering RS build their user and item profiles *implicitly*, that means by automatic feature extraction, while the class of knowledge-based recommendation algorithms include algorithms *explicitly* modeling users and items by deriving the models from the structured representation of the items. In this sense, we will focus on the feature extraction aspect of CBF, in this subsection. Again we will only give a brief overview on the basic concepts of this type of filtering technique, as it is only relevant on an abstract level for the implementation of the RS of our work.

First we will give a generally define content-based filtering, afterwards give a brief overview on algorithmic approaches for realizing CBF and finally discuss the benefits and shortcomings of CBF RS in different problem domains.

Definition of Content-based Filtering

Due to the diffuse distinction of content-based filtering and knowledge-based recommendation, in the literature there are few definitions of CBF being both compact and precise at the same time. We will consider CBF by the following comprehensive definition by Bobadilla et al. [9] in accordance to our narrow understanding of CBFs:

> "*Content-based filtering makes recommendations based on user choices made in the past [...]. Content-based filtering also generates recommendations using the content from objects intended for recommendation; therefore, certain content can be analyzed, like text, images and sound. From this analysis, a similarity can be established between objects as the basis for recommending items similar to items [...]that a user ranked positively.*"

We reduce the formal definition of content-based filtering to a minimum here, as these systems are included in the formal definition of knowledge-based recommendation found in subsection 2.3.4. The only difference is that for CBF algorithms a feature extraction function is applied on the contents of the items beforehand. This function is specified by the following formal definition:

Definition 2.18 (Space of Contents and Feature Extraction Function) *May the space of item contents $\mathscr{I}^c$ be given by $\mathscr{I}^c = A^n$, where A is an arbitrary set of values and $n \in \mathbb{N}$ is an arbitrary, but fixed number.*

Then we define the feature extraction function featex as featex : $\mathscr{I}^c \to \mathscr{I}$, where $\mathscr{I} = F_1 \times ... \times F_m$ for some fixed $m \in \mathbb{N}$ and for all $i \in \{1, ..., m\}$ it holds that F_i is a finite set of feature values. That means, featex is the mapping of the description of item contents to the space of items each specified by a finite vector of features.

In the following we will investigate, which possible implementations of content-based filtering exist by having a look at the essential parameters of those systems. These parameters are the item features and their extraction. These aspects are what we call *content representation* and the algorithms generating a user profile and exploiting it for recommendations.

Content Representation and CBF Algorithms

As the research on content-based filtering has made most advances on the topic of item contents in the text domain [31] we will refer to this for discussing feature extraction in CBF. Here commonly a given text is reduced to a set of key words (sometimes referred to as *bag of words*) or a vector of binary values indicating the presence or absence of a certain word in the text which should represent the meaning of this item's contents [40].

Regardless of the domain of the item contents the result of the extraction process is always a vector of features. We say that the items are represented inside the *vector space model* (VSM) [20]. Upon this model a plethora of CBF recommendation algorithms is defined, of which we will introduce the most common ones in the following. We stick to Janach et al. [31] and group the set of CBF algorithms into similarity-based ones and those relying on probabilistic user models.

Similarity-based CBF algorithms are strongly influenced by the field of IR and rely on the idea of finding the most similar items to the preferred ones according to their feature vector.

k-Nearest-Neighbor-Algorithm Like in item-based collaborative filtering algorithms for each unrated item a neighborhood of most similar items already rated by the current user is determined. Yet the definition of similarity for CBF algorithms relies on the feature vectors deduced from the item contents. From the set of similar items again a weighted average or a voting scheme based on the degrees of similarity is realized to determine the relevance of the current, unseen item [31]. For texts the vectors are pre-processed to receive normalized vectors, where each

component is given by a so-called *tf-idf-weight*. Furthermore, usually cosine-similarity is applied [40]. For more details refer to [40] or [31].

Rocchio's Method and Adapted Rocchio's Method It is proposed to employ the Adapted Rocchio's Method defined upon the vector space model [20]. Intuitively speaking, this algorithm derives a prototypical item vector from the rated items, which is close to the most preferred ones and far away from the ones rated badly. This explicit user model can then be used in its non-adapted form to offset a user query to return results most probably liked by the user or in its adapted version to recommend items closest to the prototypical item [31].

The set of probabilistic CBF algorithms is strongly influenced by the field of machine learning and relies on implementing classification or regression schemes trained with the pairs of feature vectors and ratings by the user.

Naïve Bayes Classifier The Naïve Bayes classifier (as described in 2.1.3) is a common method for text classification where it is usually implemented in its multivariate, Bernoulli form [32]. Using binary ratings ("relevant" and "irrelevant") as labels for the items described in the VSM a text classifier can be trained and used to recommend relevant items. Although the assumption of independence is not valid for most domains Naïve Bayes classifiers offer a solid performance [20].

Decision Trees and Rule Induction Further techniques found in machine learning are decision tree classifiers or classifiers based on rule induction. Decision tree classifiers build decision trees based on the values of the feature vectors, while rule induction classifiers deduce formal rules describing the relation of feature values [20]. Both types have been successfully applied for the recommendation problem, while leading to modest results for CBF due to their inadequacy for the text classification scenario [31]. However, it was shown that rule induction classifiers perform very well for recommendations on item feature vectors with few, meaningful components, like it can be found in many knowledge-based RS with explicit item attributes, [31]. Typical algorithms implementing decision tree classification are *ID3* and *C4.5* [31], while for rule induction a common one is the *RIPPER algorithm* [40].

Other Classifiers In the literature several linear classifiers besides the mentioned Naïve Bayes and decision trees classifiers are used to solve the recommendation problem for CBF. For the sake of completeness we give a list of algorithms employed in CBF systems without any further details: Pazzani et al. [40] suggest the *Widrow-Hoff-algorithm* or refer to systems based on the *exponentiated gradient algorithm* or *support vector machines*. Alternatively *artificial neural networks* or *Bayesian networks* have already been applied successfully for generating recommendations in CBF RS [57]. For more details refer to [40], [57] or [20].

It is noteworthy that the described algorithms can be transferred for the more general use in knowledge-based recommender systems and mostly do not rely on textual or unstructured item descriptions, but on a feature vector representation. Therefore we will refer to the set of possible, implementing algorithms of the above two groups while discussing possible approaches for modeling users in KBR and the implementation of the RS of this work (see subsection 3.3.5).

Strengths and Weaknesses of Content-based Filtering

All of the above described implementations of content-based filtering algorithms share a similar set of strengths and weaknesses. These are inherently connected to the idea of recommending items only by their extracted features without considering any other users. For the following description we will refer to many of the challenges and problems in RS which are listed in subsection 2.2.7.

First of all, content-based filtering has the advantage of generating recommendations for the current user solely based on his profile and not on the community of users [20]. This means that the quality of recommendations increases for each user depending on the amount of information known about him offering him more control and reliable recommendations regardless of the system's context.
Regarding the sparsity of user ratings, CBF posses the strength of being able to recommend new items regardless of whether any ratings by any user was given for that items. Therefore the new item problem is not given for those systems [20]. With respect to the gray sheep problem it becomes apparent that the CBF approach has the benefit of building individual user profiles, specializing on each single user [1]. Hereby every unusual user's tastes are learned individually enabling very specific recommendations, rendering the gray sheep problem of no relevance for those systems.
Additionally it is noteworthy that CBF systems, by our strict definition, do not rely on explicit meta-data manually given, but rather automatic extraction mechanisms. This can be significantly less costly from an economic perspective compared to manual content annotation. Another important aspect of content-based filtering is that it grants high scalability with regard to the size of the user base, because the recommendation generation only depends on the size of the user profile, the items description and the size of the item set.
Lastly, the user profiles in CBF RS can be mostly transferred to a human-understandable format and hereby form the

basis for offering explanations for recommended items [20]. This form of transparency is a valuable property from an application-oriented perspective.

Having a look at the downside of the content-based filtering technique, it becomes obvious that the sparsity problem in its manifestation as the new user problem persists in such systems [31]. As a new user has not rated any items his initial profile cannot reflect his interests at first and the system will take sometime until it can offer valuable recommendations to him. Further on, CBF recommender systems tend to overspecialize on the users as they build profiles exploiting them to recommend items with most similar contents to the seen ones [1]. This naturally leads to the proposition of items very similar to the ones previously liked, hereby offering no serendipities to the users. Recommending completely different items, however, can be a core aspect of a good RS depending on the application domain.
Directly connected to this problem is the phenomenon that the item space coverage of CBF RS can be comparatively low. Despite the fact that new items could theoretically be recommended, one can argue, every user is limited to a small sub-set of recommendable items according to his profile.
One of the most intensively discussed problems in CBF recommender systems is that of the limited content-analysis [20]: To generate good recommendations based on the items' contents they have to be described by useful, expressive features extracted automatically. This process, however, is very difficult for unstructured item descriptions in multimedia domains including for example image, video and audio files. For several text domains this problem cannot be dealt with, either. Here community knowledge implicitly given by the item-user-matrix is neglected, which could be used to improve recommendations with regard to this problem. The result is that items perceived as similar by the users are not identified as similar.
Additionally, from an economic perspective, the development of content-based recommender systems can be very costly: As a result of the domain dependency previous approaches in this field of research are difficult to compare and directly transfer to a new item domain [31]. Hence costly domain experts and feature engineers are necessary to design only a baseline system for a specific domain.

We draw a final conclusion on the strengths and weaknesses of CBF recommender systems from the above mentioned points: Content-based filtering has the great advantage of being able to recommend new items, propose items according to unique users' idiosyncracies and extract meaningful item features automatically. However, content-based RS tend to recommend only "more of the same", are not able to deal with new users appropriately and often suffer from insufficient feature extraction on complex item contents.

2.3.4 Knowledge-based Recommendation

The set of knowledge-based recommendation algorithms can be grouped in algorithms centrally relying on building user profiles from past feedback and those recommending through the explicit acquisition of ad-hoc user profiles usually directly specified by the user. We will refer to the first group as *user-profile-learning-based* and to the latter group as *ad-hoc-user-profile-based*. The term knowledge-based recommendation is, however, historically considered as the designator for only ad-hoc-user-profile-based RS using additional knowledge that goes beyond a history of feedbacks [23]. Nevertheless, we will consider KBR as the umbrella term for the above mentioned two groups including content-based filtering for the reasons discussed in subsection 2.3.1.

In the following we will first introduce knowledge-based recommendation informally and formally, afterwards focus on the set of KBR approaches building ad-hoc profiles of users, discuss to what degree the algorithms of CBF described in the previous subsection can be generalized for KBR and finally discuss the strengths and weaknesses of knowledge-based recommender systems.

Definition of Knowledge-based Recommendation

As their exist many views on knowledge-based recommendation it is a difficult task to give a comprehensive definition of it. One very general definition of KBR recommender systems defines them as all RS which do not employ CF or CBF techniques [23], an alternative, very narrow definition by Janach et al. [31] limits KBR to algorithms filtering item sets via explicitly given constraints (like in IR) and so-called FindMe-systems [14]. We will, however, understand KBR by the following, relatively broad definition by Ricci et al. [48], which can be found in similar forms in [20] or [54].

> "*Knowledge-based systems recommend items based on specific domain knowledge about how certain item features meet users' needs and preferences and, ultimately, how the item is useful for the user.*"

By this definition both RS building ad-hoc user profiles and RS learning user profiles over a history of feedbacks are included, as they result in a structured representation of users' needs. In this sense, we formally specify the item and user space for KBR as sets of feature vectors and leave the function extracting the user profile underspecified.

Definition 2.19 (Item and User Space for KBR) *Let $n, m \in \mathbb{N} \setminus \{0\}$ be arbitrary, but fixed natural numbers greater than zero. Then we assume the item space for knowledge-based recommendation to be of the form*

$$\mathscr{I} = A_1 \times ...A_n, \text{ where } \forall i \in \{1, ..., n\} : A_i \text{ is an arbitrary, but finite set of values}$$

Further on we assume the user space for knowledge-based recommendation to be of the form

$$\mathscr{U} = F_1 \times ...F_m, \text{ where } \forall i \in \{1, ..., m\} : F_i \text{ is an arbitrary, but finite set of values}$$

Definition 2.20 (User Profile Extraction Function) *Let $\mathscr{U}$ be the user space as specified for KBR and $\mathscr{U}^C$ be an arbitrary set of user descriptions.*

Then we define the user profile extraction function userex as the mapping from the user descriptions to the user space. Formally that is as $userex : \mathscr{U}^C \rightarrow \mathscr{U}$.

In the following we will introduce the two classes of KBR systems, constraint- and case-based recommender systems, which both employ ad-hoc user profiling. As the case-based recommender systems are central to the design of the RS of this work, we will elaborate the sub-type of *conversational recommender systems* in the isolated section 2.4. Afterwards we will discuss user-profile-learning-based KBR RS and finally outline the pros and cons of KBR systems in general.

Constraint-based Recommendation

Ad-hoc-user-profile-based KBR recommender systems rely on an explicitly given user need, like for example a keyword query or a set of preferred attribute values for the items [20]. For constraint-based recommendation this explicitly given need is specified as a set of rules for filtering the set of all available items [31].

The following explanations on constraint-based recommender systems are directly derived from the comprehensive introduction and analysis of those systems by Janach et al. [31] and Felfernig et al. [24].
In usual constraint-based RS the user specifies a set of constraints in the form of desired feature values or value ranges defining which item he is interested in. Theses constraints are in turn used to define a *constraint satisfaction problem* as defined by Janach et al. There exist various algorithms to solve constraint satisfaction problems.
From the nature of the information filtering scenario, in which recommender systems take place, follows that the user is assumed to lack deep knowledge of the domain and his own needs. Therefore constraint-based RS are usually embedded inside a sequence of interactions where the user adapts the constraints according to his needs in response to the presented items. Here the concepts of *dealing with unsatisfiable constraints*, *default values* and *value elicitation dialogs* for constraints are important directions of research to accelerate the interaction process. We will not further elaborate the mentioned terms and keep our explanations on an abstract, compact level, as constrained-based RS are not relevant for the implementation of our work. For more details refer to the above mentioned introductions to constrained-based RS.

Case-based Recommendation

Contrary to constraint-based recommendation lies the concept of case-based recommendation. Here the ad-hoc user profile is interpreted as a problem description and through similarity-based retrieval from the set of available items (in that sense, the cases) the best fitting solution to this problem is determined [48]. This approach from the field of machine learning is called *case-based reasoning* (CBR) [33].

The case-based RS are strongly connected to a conversational style of needs elicitation (refer to subsection 2.2.8 for a complete definition), where through a dialog of proposals and user feedback the recommended item or set of items is refined [11]. The most popular and deeply researched approach for realizing such dialogs is called *critiquing* or *tweaking*. On an intuitive level, critiquing recommender systems propose an initial item based on a given user query and repeatedly await a so-called *critique* by the user, followed by the retrieval of a new item most similar to the current one, yet matching the critique. This process stops when the user is satisfied by the proposed item. The (user) critique is a statement specifying for one or more item features, which value relative to the current proposed one should be assumed [34]. The semantics of a critique from the user's perspective is of the form of the following example from the domain of computers: "Show me a computer similar to the current proposed one, where the price (feature) is lower (comparator) than for the current one". This allows the user both to navigate through the item space efficiently and elicit his needs during this discovery process [31].

We will introduce the set of critiquing approaches in section 2.4 on conversational RS with more detail and in a formal manner. For now we will only give a brief, intuitive overview on implementing algorithms of the above described concept.

In their very elementary form critiquing RSs always offer the same set of critiques to select from (*static critiquing*) to the user. This however can lead to very long refinement processes. A more sophisticated approaches is *dynamic critiquing* where the critiques the user can select are determined from the features of the items left in the case-base. Dynamically generated critiques on multiple features are so-called *compound critiques*. These compound critiques are usually deduced from regularities of the features of the underlying item set relative to the currently proposed one.

User Profile Learning in Knowledge-based Recommender Systems

In subsection 2.3.3 we introduced multiple algorithms implementing the content-based filtering technique which come from the field of IR and machine learning. It is noteworthy that, in principle, these approaches can be transferred to the larger set of knowledge-based recommender systems. In the following we will give a brief discussion on transferring user profile learning algorithms to items with explicitly defined features.

Although several algorithms for CBF are discussed and implemented with respect to textual item contents in the literature, all rely on the vector space model representation of the items, that means by feature vectors. As this is the general representation of items in knowledge-based recommender systems the basic algorithmic approaches can be transferred with minor modifications for being able to deal with different value domains of the items' features. As a matter of fact, Pazzani et al. [40] and Zukerman et al. [57] introduce possible algorithms already coined for the more general set of recommender systems relying on item descriptions, which we call knowledge-based RS.

Strengths and Weaknesses of Knowledge-based Recommendation

After the above overview on the two types of KBR we will now discuss their strengths and weaknesses referring to common problems in recommender systems as specified in subsection 2.2.7. For the sake of clarity we will differentiate between ad-hoc-user-profile-based and user-profile-learning-based KBR and therefore treat the benefits and shortcomings of these types one after another.

Probably the greatest benefit of KBR RS which build ad-hoc user profiles is that there is no ramp up phase for new users [31]. As every user defines his preferences anew per session, he can receive valuable recommendations fitting his current needs; therefore the new user problem can be successfully dealt with in such systems. This is especially relevant in domains, like for example shopping for cars, flights or technical devices, where the number of user feedbacks on items is generally low or spread over very long periods of time.
Another important aspect is that ad-hoc user profiling KBR recommender systems are suitable for domains of items with complex feature vectors where the problem of information overload already arises by the, from a user's perspective, overwhelmingly complex item descriptions. Here such systems, especially conversational ones, can offer a great way to reduce the complexity for the user [11].

With regard to the weaknesses of ad-hoc-user-profile-based KBR RS one central aspect is, of course, that those system require additional effort of the user compared to approaches based on transparently building profiles and making (proactive) recommendations. Additionally the knowledge engineering process during the design of the system is often costly and error prone [54]: For certain item sets the domain given features do not sufficiently describe the items from a user perspective or the underlying similarity measure (for case-based approaches) does not reflect the general, human view. For example, while buying furniture on an e-commerce platform the length, height and depth of an cupboard might be central decision criteria for the user, but in the end more subtle and subjective properties like the appearance will play an important role. This can be viewed as an instance of the problem of limited content analysis.

For user-profile-learning-based KBR recommender systems basically the same strengths and weaknesses as for content-based filtering arise: They can recommend new items and propose items according to individual tastes, while often recommending "more of the same" and not being able to deal with new users. However, the problem of limited content analysis of CBF RS does not necessarily apply for the general type of knowledge-based recommender systems. For all approaches realizing KBR with learning of user profiles, of course except for CBF, the structured item features are intrinsically given by the application domain. However, as described above for ad-hoc-user-profile-based systems, the knowledge engineering process while designing the feature space and similarity measures of the items, can lead to inaccurate item representations, as well.

We draw a final conclusion on the strengths and weaknesses of KBR recommender systems from the above mentioned points: Ad-hoc-user-profile-based systems are highly suitable for item domains with complex features and with very few and obsolete feedback data for each user. They can overcome the new user problem, but impose the user with additional effort. User-profile-learning-based RS posses very similar strengths and weaknesses as the subclass of CBF described in subsection 2.3.3.

2.3.5 Demographic Recommendation

The research on demographic recommendation has made only little advances over the last decades [48]. Therefore we will only give a brief overview on the basic concept without going into the details by discussing possible, implementing algorithms.

Definition of Demographic Recommendation

Due to the negligence of this filtering technique in the mainstream of RS research there exist only few definitions of demographic recommendation in the introductory literature. As a matter of fact, some authors understand demographic recommendation as a form of knowledge-based recommendation because such systems take additional knowledge beyond a history of user feedbacks into account [23]. We will understand demographic recommendation by the following definition by de Gemmis et al. [20]:

> "*These systems [demographic recommender systems]aim to categorize the user starting from personal attributes making recommendation based on demographic classes.*"

We will spare the formal definition of DR, because those systems can be modeled similar to KBR (refer to subsection 2.3.4).

The most obvious approach for implementing demographic recommendation is by handcrafted rules from the domain knowledge of which demographic groups prefer which items. More sophisticated approaches relying on data mining and machine learning techniques basically extract those rules automatically [20].

Strengths and Weaknesses of Demographic Recommendation

From the given abstract description it is a difficult task to derive strengths and weaknesses of demographic recommendation in a reliable manner. Nevertheless we will give an overview on the general, obvious pros and cons following directly from the definition of those systems.

The central advantage of demographic recommender systems is that they successfully deal with the new user problem: There is no ramp-up phase for newly registered users, as they usually specify the demographic information once at the beginning and can already receive valuable recommendations using this profile [20].

Having a look at the downsides of demographic recommendation it becomes apparent that deep domain knowledge is required to receive a good performing system that goes beyond proposing the same, most favorite items to all users [20]. Both for manually and automatically derived rules a big data set of users and their (implicit) item ratings in the past are necessary to determine well-founded sets of rules.

2.3.6 Hybrid Recommender Systems

From the strengths and weaknesses of the previously introduced filtering techniques it becomes obvious that each approach is only focused onto a subset of the actually available information in recommender systems. This inherently limits the quality of recommendations in different application scenarios to a certain low level. Naturally the idea of combining multiple algorithms of different filtering techniques comes to mind. This is subject of designing hybrid recommender systems, which we will investigate in the following.

First of all, we will give an informal and formal definition of hybrid recommender systems and afterwards list hybridization approaches found in the literature. This list forms the basis for a practical guideline on choosing a hybridization design for a given problem.

Definition of Hybrid Recommender Systems

The following definition of hybrid recommender systems is extracted from the milestone paper on hybridization by Burke [13], which most papers in this field of research refer to. For this reason we will consider HRS by this definition despite its quite general nature.

> "*Hybrid recommender systems combine two or more recommendation techniques to gain better performance with fewer of the drawbacks of any individual one.*"

From this definition we derive the following, formal model of hybrid recommendation algorithms leaving certain parts underspecified. Note that the formalization only includes binary combinations of RS, but by the recursive application of the definition allows more complex RSs.

Definition 2.21 (Hybrid Recommendation Algorithm) *Let $\mathscr{R}$ be the space of all recommendation algorithms (i.e. algorithms solving the recommendation, prediction or ranking problem for some item and user domains).*

Then we call a mapping h, where $h : \mathscr{R}^2 \to \mathscr{R} : (r_1, r_2) \mapsto r_{res}$ an hybridization function and call the image r_{res} the hybrid recommendation algorithm of r_1 and r_2 by h.

From a formal perspective, the following list of techniques for combining recommendation algorithms of the different filtering techniques specify the hybridization function of the above definition. Therefore we will refer to it for a more precise definition of the techniques.

Design Space of Hybrid Recommender Systems

The following list of seven elementary hybridization techniques is based on the comprehensive analysis of hybrid recommender systems by Burke [13]. The organization of them in *parallel*, *monolithic* and *pipelined* hybridization designs, however, was introduced by Janach et al. [31]. For an enhanced clarity we will stick to these classes of basic designs and elaborate them in the following. The whole list can be considered as the design space for hybrid recommender systems and consequently may be used as a guideline for designing HRS.

For the sake of precision we will add the formal specification for each technique using the above definition of hybrid recommendation algorithms. For this purpose let r_1 and r_2 name the input recommendation algorithms and $r_{res} = h(r_1, r_2)$ the result of the hybridization. For every algorithm we associate an item and user space, as well as the approximated utility function and the set of recommended items per user. We refer to them by $\mathscr{I}_i$, $\mathscr{U}_i$, $u_i : \mathscr{I}_i \times \mathscr{U}_i \to R$ and $recs_i : \mathscr{U} \to \mathscr{P}(\mathscr{I}_i)$ for the adequate $i \in \{1, 2, res\}$.

Parallel Hybrid Recommender Systems

Parallel hybrid recommender systems employ the two recommendation algorithms on the same item and user space, that means in parallel. Their results are afterwards combined by some algorithm. Hence, from a formal perspective, we can always assume $\mathscr{I}_1 = \mathscr{I}_2 = \mathscr{I}_{res}$ and $\mathscr{U}_1 = \mathscr{U}_2 = \mathscr{U}_{res}$. Janach et al. understand the following three hybridization techniques as an instance of this design.

Weighted Hybrid In this type of HRS the predicted scores or generated recommendations of the underlying recommendation algorithms are combined via some weighing scheme. For predicted scores a weighted sum is formed, while for sets of recommended items voting schemes can be implemented. Formally for the case of predicted scores the following specification of the resulting utility function estimation is received:

$$u_{res}(x, y) = \alpha_1 \cdot u_1(x, y) + \alpha_2 \cdot u_2(x, y)$$

For the case of combining sets of recommendation results, hence using a voting scheme, we can specify the resulting utility function estimation as follows:

$$u_{res}(x, y) = \alpha_1 \cdot \begin{cases} 1, & \text{if } x \in recs_1(y) \\ 0, & \text{else} \end{cases} + \alpha_2 \cdot \begin{cases} 1, & \text{if } x \in recs_2(y) \\ 0, & \text{else} \end{cases}$$

For both equations it holds that $\alpha_1, \alpha_2 \in \mathbb{R} \setminus \{0\}$.

Switching Hybrid Switching hybrid recommender systems predict the score for one item always using only the score of one of the two recommendation algorithms. A switching mechanism is employed, which is for example connected to the level of confidence of the first recommendation algorithm (maybe measured by the underlying number of previous ratings) or the current session context (maybe given by the features of previously consumed items). This ensures that the adequate recommendation algorithm is used in each situation, while also introducing complexity. Formally we model the resulting utility function, as follows:

$$u_{res}(x, y) = \begin{cases} u_1(x, y), & \text{if } c = \text{true} \\ u_2(x, y), & \text{else} \end{cases}$$

Here c is an underspecified propositional formula modeling the switching condition.

Mixed Hybrid In mixed hybrid recommender systems the resulting sets of recommendations are blended into one set, or rather for resulting rankings the ordered lists are combined. This can take the form of either additive merging of the recommendations or the parallel presentation of the results. This approach should, by tendency, be only applied if the cardinality of the set of generated recommendations is not strongly limited. From a formal perspective we may model the mixing of recommendation results by simply uniting the sets, which induces an infinite number of utility functions resulting in this set of recommended items. In the following we define the trivial utility function assigning the same score to all included items.

$$u_{res}(x,y) = \begin{cases} 1, & \text{if } x \in recs_1(y) \vee x \in recs_2(y) \\ 0, & \text{else} \end{cases}$$

$$\forall y \in \mathcal{U} : recs_{res}(y) = recs_1(y) \cup recs_2(y)$$

The case of combining ranks is a special case of the one above, where the definition of the utility function is narrowed down by some rank combination algorithm. We will not explicitly model this here.

Monolithic Hybrid Recommender Systems

Monolithic hybrid RS combine the two underlying recommendation algorithms, their data and concepts in a way that together they cannot be assigned a single filtering technique presented in the previous subsections. The resulting RS consists of only one recommendation algorithm at its core.

Feature Combination Hybrid HRS relying on feature combination basically do not combine two distinct recommendation algorithms, but rather their underlying, raw item and user descriptions. Concretely this means that the item and user features used in the first algorithm are added to the feature vectors of the items and users of the second one. Then a new recommendation algorithm is defined upon those features, usually derived from both or one of the initial algorithms given. Formally we model the resulting feature spaces as the Cartesian product of the input spaces, that means $\mathcal{I}_{res} = \mathcal{I}_1 \times \mathcal{I}_2$ and $\mathcal{U}_{res} = \mathcal{U}_1 \times \mathcal{U}_2$. The utility function u_{res} of the resulting algorithm is arbitrarily estimated; it is usually, however, similar to one of the initial algorithms.

Feature Augmentation Hybrid In hybrid recommender systems using feature augmentation the feature space of the first algorithm is extended by the results of the second one. This differs from feature combination in the fact that there are two recommendation algorithms executed, while the results of the first ones are treated as features of items or users for the second one. Here the extended item space is the result of a complex processing step unlike in feature combining hybrids. But in the end only one recommendation algorithm is applied for solving the recommendation problem. We formally model $\mathcal{I}_{res} = \mathcal{I}_2 \times P_I$ and $\mathcal{U}_{res} = \mathcal{U}_2 \times P_U$, where P_I, P_U are arbitrary feature spaces derived from the partial results and computations of r_1. The utility function is arbitrarily estimated.

Pipelined Hybrid Recommender Systems

Pipelined HRS rely on the solution of the recommendation or prediction problem by one of the algorithms and use the resulting scores or sets of recommended items as an input for the second recommendation algorithm. This can be viewed as an refinement process, where every recommendation component can be separated from the other, unlike in monolithic HRS. Here we formally assume $\mathcal{I}_1 = \mathcal{I}_2 = \mathcal{I}_{res}$ and $\mathcal{U}_1 = \mathcal{U}_2 = \mathcal{U}_{res}$.

Cascade Hybrid In cascade hybrid recommender systems the first recommendation algorithm determines a set of potentially interesting items and the second one ranks and selects this item set, hereby refining the recommendation of the first. Thereby we receive the following utility function:

$$u_{res}(x,y) = \begin{cases} u_2(x,y), & \text{if } x \in recs_1(y) \\ 0, & \text{else} \end{cases}$$

Meta-level Hybrid Meta-level HRS are similar to feature augmentation and combination hybrids. They pass the whole user profile built by the first recommendation component as a parameter to the second recommendation algorithm. Hereby they differ in the fact that complex user models are passed, which go beyond simple features. Therefore we introduce a parameter m_1 representing the model built by the first recommender and leave the utility function $u_{res}^{m_1}(x,y)$ underspecified.

By the above list of design options it becomes apparent that there exists a variety of ways of implementing hybrid recommender systems. The general tendency of HRS is to improve performance as the shortcomings of the single filtering techniques can be compensated through combination. Nevertheless, a hybrid design increases the number of parameters to adjust making the development process more difficult.

2.4 Conversational Recommender Systems

In this section we will focus on conversational recommender systems relying on an interactive dialog with the user to elicit his current preferences. Although this technique comes from the field of knowledge-based recommender systems there already exist various hybrid implementations relying on other filtering techniques. This form of RS is of great relevance as the implemented recommender system of our work is used inside a chat-bot application; hereby a dialog-style of refining recommendations is a very natural approach for this RS.

In this section we will first give a definition of conversational recommender systems and discuss possible approaches for their implementation. In the rest of the section we will focus on the approach of critiquing-based RS with respect to its definition and concepts. Later we will have a look at challenges and implementations of critiquing-based RS.

2.4.1 Definition or Conversational Recommender Systems

The field of research on conversational RS consists of many different approaches for learning, integrating and presenting user preferences. In this subsection we will give a general definition and classification of those systems and afterwards introduce several classes as a taxonomy for ConvRS.

Definition and Classification of Conversational Recommender Systems

There exist several definitions of *conversational recommender systems* (ConvRSs) connecting them directly to dialogs in human language (e.g. in [26]), which is, from our perspective, misleading and vague. The following brief definition by Bridge et al. [11] gives a more abstract understanding of ConvRSs as systems relying on user-interactive result refinement procedures:

> "*Conversational recommenders adopt an iterative approach to recommendation. Users can elaborate their requirements, as part of an extended recommendation dialog.*"

We will now give an attempt in classifying conversational recommender systems by the taxonomy of filtering techniques (refer to subsection 2.3.1). By the above definition the elicitation of short-term preferences, that means needs expressed in the current recommendation session, is a central aspect of ConvRSs. Typically this is done via explicit user and item models by domain features rendering conversational RS to be of the type of ad-hoc-user-profile-based KBR. Nevertheless, there exist hybrid implementations employing for example collaborative filtering (as in [43]), content-based filtering (as in [12]) or KBR with the learning of user-profiles (as in [26]). Therefore we propose to classify conversational recommender systems as forms of hybrid recommender systems, where at least one component is knowledge-based and employs ad-hoc generated user profiles.
In the rest of this subsection we will refer to this knowledge-based component when we talk about ConvRSs.

Taxonomy of Conversational Recommender Systems

Conversational recommender systems can be reasonably divided into two classes by the amount of information presented to the user during the interactive dialog [11]: *Navigation-by-asking* and *navigation-by-proposing* ConvRS. In navigation-by-asking-based systems the user is presented a sequence of questions derived from the item catalog to reduce the set of potentially relevant items iteratively. Yet he is never presented a concrete item until the very end of the dialog. Contrary to this approach navigation-by-proposing-based RS present an item in each iteration and ask for feedback on the properties of this item by the user.

The set of navigation-by-proposing ConvRS is usually further differentiated by their type of feedback requested from the user. *Preference-based feedback* is a form of feedback where the user is presented multiple items (at least two) and responds with the selection of one of the items, hereby defining a preference relation on them. On the other hand, *feedback as a critique* (or simply *critiquing*) is used in a scenario where the user is presented one item and gives a feedback on how he would alter one ore more features of the given item to meet his needs better [11]. This differentiation results in the hierarchic taxonomy of conversational RS as depicted in figure 2.3.

Navigation-by-asking RS suffer from the intrinsic weakness of requiring the user to have a quite clear vision of his most preferred item and to posses relatively deep domain knowledge [35]. As, depending on the approach, even concrete value definitions for features are required from the user,s rendering it rather an information retrieval scenario than an information filtering scenario, in which recommender systems are actually located. Therefore most research has focused on RS using proposing, which we, as well, will discuss in its form of critiquing-based conversational RS, promising the best trade-off between user effort and low ambiguity of feedback [35], in the following subsections with more detail.

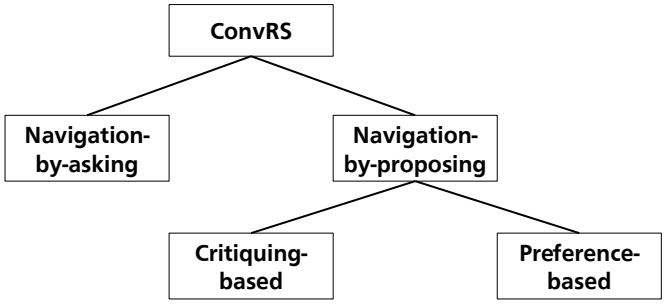


Figure 2.3.: The taxonomy of conversational recommender systems depicted as a tree graph. The nodes represent system configurations beginning with the most abstract description of ConvRS in the root to the least abstract ones in the leaves.

2.4.2 Definition of Critiquing-based Recommender Systems

Critiquing-based (or critiquing) recommender systems (CBRS) are the most popular and deeply researched form of navigation-by-proposing-based conversational recommender system. In the literature they are treated as one of the best approaches for solving the recommendation problem efficiently in complex item domains for users with both little and deep domain knowledge [34].

In the following we will first explain how the general flow of dialogs in critiquing recommender systems looks like, derive a formal model from it and introduce various types of critiquing RS as a taxonomy for those systems.

Informal Definition of Critiquing-based Recommender Systems

Chen et al. [16] define an interaction model for critiquing RS onto which we lean the following, more general one. In general CBRS are reactive RS (see 2.2.8) and refine the recommendation, just like all conversational RS, until either the user aborts the process or accepts the recommended item. Hereby we have the following iterative steps in an critiquing dialog starting at an initial user query and ending in either an aborted session or an accepted item.

1. The user defines a query and sets-off the recommendation process.
2. The system proposes an item based on the given query, previous feedback and a-priori knowledge on the user.
3. The user decides on either aborting the session, accepting the item or further refining.
4. A set of critiques referring to the presented item is offered to the user.
5. The user reviews the presented item and defines a critique usually by selecting it from the set of offered critiques.
6. The critique is applied and the next item to be presented is determined.
7. The steps 2. to 6. are repeated until either the user accepts the item or aborts the session.

This model of dialogs in critiquing-based RS is illustrated by an decision-activity-diagram in figure 2.4. It is noteworthy that usually the sixth step is realized by choosing the most similar item (according to some measure) matching the critique. In this selection process, however, many other information can be considered. The above description of dialogs in CBRS serves as an informal definition of critiquing-based RS from which we derive a formal model in the following paragraphs.

Terminology of Critiquing-based Recommender Systems

For the formalization of CBRS and an discussion of them on an adequate level we will now introduce common terms for the concepts described by the above model of critiquing dialogs. The following terms and descriptions are a conclusion of the definitions in [34].

One instance of an incremental refining dialog is called a *(recommendation) session*. The iterations of such a session are called *(recommendation) cycles*. A critique that refers to one single feature is called a *unit critique*, while one referring to at least two features is called a *compound critique*. Usually unit critiques are differentiated by the feature domains upon which they are defined. Critiques on numerical domains asking for an increase or decrease in the value of the feature are called *directional critiques*, while those on arbitrary domains restricting the value of the feature to assume any value, but the current one are *replacement critiques*. Finally, the set of critiques offered to the user is called *critique options*.

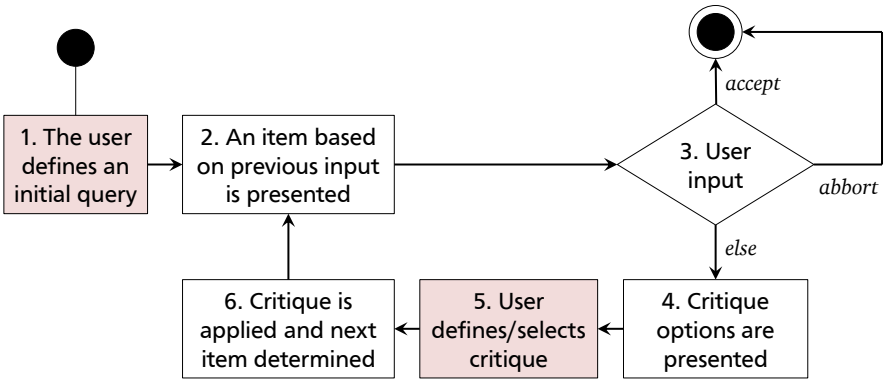


Figure 2.4.: The model of interactive dialogs in critiquing-based recommender systems depicted as an decision-activity-diagram. The activities where the user is involved are marked in light red.

Formal Model of Critiquing-based Recommender Systems

So far we have defined critiquing-based recommender systems informally by their dialog behavior. Hence a natural approach for designing a precise, formal model is to define them by a generic algorithm.
The user and item space for CBRS are given by the definition of knowledge-based recommendation (refer to subsection 2.3.4). We then define a unit critique to be a predicate function from the domain of the respective feature to the boolean values. While a compound critique can be specified as the application of more than one unit critique on their respective feature spaces. For an easier representation we will not formally limit compound critiques to more than one unit critique.

Definition 2.22 (Unit Critique) *Having the item base given like in definition 2.19, i.e. for* $n \in \mathbb{N} \setminus \{0\}$ *the itemset* $\mathscr{I} = A_1 \times ... \times A_n$*, where* $\forall i \in \{1,...,n\} : A_i$ *is an arbitrary, but finite set of values.*

Then a mapping $c_i : A_i \to \mathbb{B}$ *for* $i \in \{1,...,n\}$ *is called a unit critique on feature* A_i*. The function* c_i *is called directional, if and only if* A_i *is numeric and the mapping* c_i *can be specified by an equation of the following form.*

$$c_i(x) = (x \circ \tau) \text{ for a fixed } \tau \in A_i \text{ and } \circ \in \{<,>\} \text{ for } < \text{ a total ordering on } A_i\text{, where } > \text{ is the inverse ordering to } <$$

The function c_i *is called a replacement critique, if and only if* c_i *can be specified by an equation of the following form.*

$$c_i(x) = (x \neq \tau) \text{ for a fixed value } \tau \in A_i$$

It is common to specify unit critiques by the triple $(f, \circ, \tau)$, where f refers to one of the attribute spaces, $\circ$ is a symbol from $\{<,>,\neq\}$ and τ is the comparison value as in the above equations. Alternatively, for a given referred item, unit critiques can be represented by only the first two components as τ assumes the item's value of the named attribute. This fully specifies a unit critique and its type, as long as the attribute domains and their orderings are given.

Definition 2.23 (Compound Critique) *Having the item base given like in definition 2.19, i.e. for* $n \in \mathbb{N} \setminus \{0\}$ *the itemset* $\mathscr{I} = A_1 \times ...A_n$*, where* $\forall i \in \{1,...,n\} : A_i$ *is an arbitrary, but finite set of values.*

Compound critiques are defined as the mappings $cc_{\{a,...,z\}} : A_a \times ... \times A_z \to \mathbb{B}$ *for* $\emptyset \neq \{a,...,z\} \subseteq \{1,...,n\}$ *where the following equation holds.*

$$cc_{\{a,...,z\}}(x) = \bigwedge_{i \in \{a,...,z\}} c_i(x), \text{ where } c_i \text{ is a unit critique on } A_i$$

Usually compound critiques are specified by the finite sequence of triples defining the unit critiques which the compound critique is made up of. Again we can reduce the unit critiques to the pair-wise representations described above, if a referred item is given. This is a complete representation of the compound critique.
Obviously from both unit critiques and compound critiques a filter function can be constructed mapping from the set of items $\mathscr{I}$to the boolean values, which we will refer to in the following. We will not elaborate this step as it is trivial.

Having these definitions in mind we can specify a generic algorithm for representing CBRS formally. For the sake of generality we will leave the sub-functions underspecified. Algorithm 2.1 can be viewed as a more general formulation of the one specified by Janach et al. [31]. It will be used to define critiquing approaches precisely.

Algorithm 2.1 The algorithm formally modeling critiquing-based recommender systems.

Require: q is the query given by the user;$\mathscr{I}$is the item base

```
1:  procedure CritiquingRecommendation(q, 𝓘)
2:      state ← "refine"
3:      curitem ← InitialItemRecommendation(q, 𝓘)
4:      curfilter ← (x ↦ true)
5:
6:      loop
7:          state ← ProposeItem(curitem)
8:
9:          if state ≠ "refine" then
10:             break
11:
12:         curopts ← GenerateCritiqueOptions(curitem, 𝓘)
13:         curcrit ← ProposeCritiqiueOptions(curitem, curopts)
14:
15:         curfilter ← UpdateFilter(curitem, curcrit, curfilter)
16:         catalog ← ApplyFilter(curfilter, 𝓘)
17:         curitem ← SelectNewItem(catalog)
18:     end loop
```

The underspecified functions in algorithm 2.1 are themselves algorithms with the following, intended functionalities: InitialItemRecommendation generates the initial recommendation from the query and item set as an input. This item is often called the *entry point* of the session. The ProposeItem algorithm presents the current item to the user and returns, whether he wants to proceed with the session, GenerateCritiqueOptions deduces a set of possible critiques from the item set and the current item, while ProposeCritiqueOptions presents them together with the referred current item and returns a critique as a response. Finally, the UpdateFilter function translates the received critique into the current filter, which is used by ApplyFilter together with the underlying item catalog to select a subset of relevant items. From this subset of items the next current item is chosen via the SelectNewItem algorithm.

It is noteworthy that by this very general definition all parts of the critiquing process can be designed considering various knowledge sources. For example for the SelectNewItem procedure typically some similarity based selection is employed, yet the initial query or past critiques could be considered for selecting the next item, as well.
In the following subsections we will refer to this generic algorithm and specify the algorithmic sub-functions according to the concrete implementation approaches in critiquing-based recommendation.

Taxonomy of Critiquing-based Recommender Systems

By the set of the seven underspecified functions defined above a general design space of CBRS was implicitly defined. This will serve us now to introduce basic critiquing approaches.

Typically the set of CBRS is described in two dimensions: The employed *type of critiques* and the *generation of critique options* [34]. For the first dimension a critiquing-based RS can either rely solely on unit critiques or employ compound critiques (maybe in combination with unit critiques). Further on the generation of critique options (GenerateCritiqueOptions in algorithm 2.1) can be realized via the trivial algorithm returning always the same set of options or alternatively by a more sophisticated algorithm deducing a set of critique options from the item catalog and the current selected item. Here the terms *static critiquing*, for the trivial algorithm, and *dynamic critiquing*, for the situation-dependently choosing algorithms, are used [45].

Hereby we receive a tabular taxonomy scheme to classify CBRS from a perspective focusing on the generation of critiques in such systems. The result is depicted in table 2.5. For every configuration of dynamic, static, unit and compound critiques there exist propositions implementing them, which are named in this table. One should be aware that in the literature the term *dynamic critiquing* is often used synonymously to what we understand as dynamic, compound critiquing [45]. Yet the concept of dynamic critique option generation can, of course, be applied to unit critiques [34]. Contrary to theses configurations the static, unit critiquing CBRS are often simply referred to as *static* or *simple critiquing* RS [31].

		Critique Generation	
		Static	**Dynamic**
Critique Type	**Unit**	Static Unit Critiquing (*Entreé* [14])	Dynamic Unit Critiquing (*TweakSystem* [36] as in [34])
	Compound	Static Compound Critiquing (*CarNavigator* [14])	Dynamic Compound Critiquing (Computer-RS in [45])

Table 2.5.: Taxonomy of critiquing-based recommender systems from a perspective focusing on the generation of critiques. Every possible CBRS is assigned a class according to the critique types employed and their generation. For each class one example proposition in the literature is named.

2.4.3 Challenges in Critiquing-based Recommender Systems

There exist various challenges in CBRSs arising from their special form of navigation through the item space. The following list of challenges or issues will serve as a guideline for evaluating the qualities of the different implementations of CBRS discussed in the next subsections.

In this subsection we will first introduce an alternative view on CBRS regarding the exploration of the item catalog and afterwards use this view to explain several of the typical problems arising in critiquing-based recommender systems.

Item Space Exploration in Critiquing-based Recommender Systems

Unlike, for example, search engines or single-shot recommender systems conversational RS like CBRS navigate through the item space giving the user insight to a subset of potentially relevant items; we call this process *(item space) exploration*.

A very useful and intuitive perspective for describing the exploration behavior of critiquing-based recommender systems is to interpret the navigation inside the item space according to the vector space model [31]. Each item can be located as a vector or point inside the space of all attribute combinations. The current item can be perceived as the origin of this space and all items are represented relative to it. Hereby unit critiques, referring to the current item, separate the space at the origin orthogonal to the axis of the attribute associated by the critique. In figure 2.5 this would mean that for example a unit critique $(A_1, >)$, which is a directional unit critique requesting a higher value in A_1, would only allow items in the first and second quadrant of the depicted coordinate system.
The sequence of critiques generated during the session, which we will call *(critiquing) path*, can then be viewed as the sequence of vectors transposing the current items to their respective next items. The critiquing path starts at the initially proposed item and terminates at the current one. A successful session therefore always connects the entry point with the booked item by a finite sequence of transpositions. This perspective on item space exploration in CBRS is illustrated for the two-dimensional case in figure 2.5.

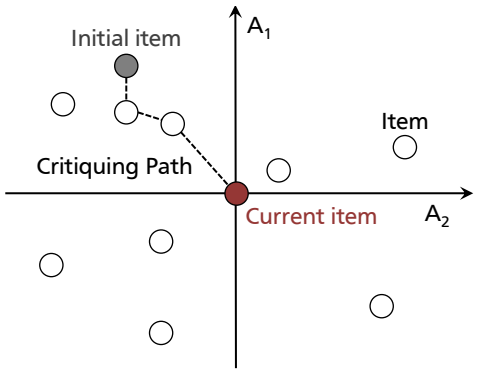


Figure 2.5.: Illustration of the item space navigation in CBRS for a two-dimensional item space. The space is represented via a Cartesian coordinate system where each axis represents the domain of one feature. The critiquing path initial and current item are highlighted.

Challenges and Issues of Erroneous Critiquing Paths

One central aspect of the quality of CBRS is efficient preference elicitation [34]. An inefficient strategy is manifested in an unnecessary great length of the recommendation session; that means the number of critiques on the critique path from the initial item to the final item is greater than necessary. This aim is directly linked to the following set of issues related to erroneous and hereby prolonged critiquing paths.

Retrieval Failures During a critiquing session a user might select a critique on the current item, which results in an empty result set, as no item matches the resulting filtering condition. This is called a *retrieval failure* [34] and requires CBRSs either to offer a repair mechanism or prevent this case by limiting the critique options.

Non-Reversibility Problem For several CBRS approaches making one step from the current item to the new one is not necessarily reversible, as e.g. the reverting critique is not offered or the resulting new item is selected context-dependent. This can lead to unnecessarily long sessions as the user may not be aware of his preferences at first, requiring him to repair it through multiple critiques [35].

Non-reachability Problem Although theoretically all items of the product space might be reachable via a sequence of critiques in a CBRS, there exist cases of cyclic navigation through context-dependent item selection in certain CBRS approaches, which lead to the circumstance that the most desired item is not reachable [34]. In such situations the user cannot be satisfied.

Inconsistency of Critiques A special instance of the non-reachability problem is the problem of *inconsistent critiques* [34]. This problem arises in CBRS which consider the past sequence of critiques besides the current one to determine the filter. Here inconsistent critiques caused by unclear needs of the user can lead to unsatisfiable filters. This has to be prevented or a repair mechanism is needed.

Challenges and Issues of Over-simplification

CBRS offer the reduction of complexity of large item spaces where every item is described by many and complex features. This, however, can lead to several problems related to the reduced, sometimes over-simplified view on the item space and the limited interactional possibilities of the user during the exploration process.

False-leads In the exploration process the user might follow a critiquing path expecting to find the ideal item at the end. This, however, is not always the case as the critique options do not offer any information on the resulting items' features which are not associated with the critique. These unnecessary critiquing paths are called *false-leads* [35] and can be associated with the broader problem of the *limited product space view* [34] of the user. This issue of increasing the understanding of the item space and the influence of each critique on the next, potential items is central to CBRS.

Slow Navigation Connected to the limited product space view, as described above, the problem of slow navigation as a result of the ambiguous nature of critiques arises [31]. In scenarios where many items match a series of critiques leading towards the ideal item, the filter derived from the critique only reduces the result space marginally. This means that there exist many items which ares similar with respect to the critiqued features relative towards the entry point. Multiple of them have to be reviewed in classical approaches until reaching the final, ideal item hereby requiring more cycles. In those dense areas of similar items the exploration process can be very slow due to the imprecise constraints implied by the critiques.

Weakly Relevant Feedback Options In the dialog process of CBRS the user is proposed a limited number of critique options in each cycle to prevent overwhelming him. Here a subset of critique options might be selected which does not allow the user to navigate according to his preferences. If, for example, for the user of an e-commerce platform money is no object, he will certainly not be interested in critiques on the price feature. McGinty et al. [34] refer to this phenomenon as *weakly relevant feedback options*, which is directly related to the broader problem of *restricted user control* [34] describing that users are limited to selecting proposed critiques. Avoiding this limitations can be an important objective of the development of CBRS for an increased user comfort.

The above described problems in critiquing-based recommender systems motivated the development of various approaches for critiquing-based recommendation, which try to tackle one or more of these issues, while at the same time suffering from other problems. We will elaborate the most important approaches in the following subsections.

2.4.4 Basic Approaches for Critiquing-based Recommender Systems

Historically the most important critiquing-based recommender systems rely on the ad-hoc generation of user profiles, like they can be found in the first so-called FindMe-systems [14]. The approaches differ in the definition of the underspecified functions of algorithm 2.1, while having in common that they construct a user profile only for the current session. We call them *basic approaches* as they are all pure (i.e. non-hybrid) knowledge-based systems with ad-hoc-user-profiling.

In this section we will first introduce the basic implementation approaches of static unit critiquing, dynamic critiquing and later the more sophisticated incremental critiquing. Finally we will give a very brief overview on alternative, recent developments which are, however, significantly less well-known and not as deeply researched.

Static Unit Critiquing

Static unit critiquing (or simple critiquing) belongs to the same-named class of CBRS in the taxonomy introduced in subsection 2.4.2. In its very essence this approach is defined via the following working principle [31]: The initial item is determined by the best match for the given user query. Exclusively unit critiques are proposed in a static manner. Only the current item is proposed together with this static set of unit critiques. The user is modeled by his currently given feedback ignoring previous critiques. The current filter simply accepts all items matching the current critique. The next item from the set of valid items is then determined by selecting the most similar one, according to a similarity measure. This is formalized by the functions defined in algorithm A.1. We skip the ProposeItem and ProposeCritiqueOptions as those are trivial user interface functions in static unit critiquing.

The obvious downsides of this approach are the tendency to retrieval failures, as the critique options are not chosen dependent on the available items, [31] and the slow navigation due to the fact that unit critiques do not restrict the solution space very strongly compared to compound critiques [45]. Additionally, it can be shown that non-reachability is an issue in static unit critiquing RS [37] (as cited in [34]).

Dynamic Critiquing

To solve the problem of inefficient navigation of static unit critiquing the concept of *dynamic critiquing* was developed. It belongs to the class of dynamic, compound critiquing according to the introduced taxonomy in subsection 2.4.2. This approach is similar to static unit critiquing except for the following two aspects [45]: Firstly not only unit critiques, but also compound critiques are offered to the user. Secondly, all critiques are dynamically derived from the set of available items relative to the currently presented one. We will elaborate on these two mechanisms in the following.

For realizing the automatic derivation of possible and most efficient compound critiques from the item space there exist two basic approaches: *Apriori-based* and *MAUT-based* systems [47]. Here efficiency means the reduction of the number of items after the application of the respective filter implied by the critique. This is usually measured in the proportion of the number of leftover items compared to the cardinality of the whole item space [31]. In the following we will give an overview on this two approaches. The description of Apriori-based systems is leaned onto [45].

The idea of Apriori-based CBRS is to employ the Apriori algorithm (as described in 2.1.2) for discovering regularities in the set of the so-called *critique patterns*. A critique pattern of one item of the catalog is defined as the vector of comparators for each feature received by comparing the feature values of the considered and the current item. For numerical domains one symbol of the set $\{>, <, =\}$ and for non-numerical ones of the set $\{\neq, =\}$ are possible comparators defined by the orders on the respective feature domains. An example for the critique patterns of a four item candidate set with two features can be found in table 2.6. On the set of these patterns the Aprori algorithm is applied, where each pattern is considered as a transaction and the comparators as what is called an item in association rule mining.

The generated rules describing relations between comparators on features, hereby each fully specifying a unit critique (see subsection 2.4.2) for the given proposed item, are then translated to compound critiques. The translation is done by joining all implied unit critiques of the antecedent and the consequent to one compound critique. Therefore only possible combinations of unit critiques are generated. They are usually ranked according to the support of their underlying rules, as minimal support implies a minimal set of items meeting this critique. This results in a ranking of critiques according to their efficiency. The formalization of this mechanism is specified in algorithm A.2. The initial item recommendation, the selection of a new item and filter update mechanisms are identical to the ones in static unit critiquing described in algorithm A.1.

	Item Description		Critique Pattern	
Item ID	**Price**	**Color**	**Price**	**Color**
curitem	120	'blue'	-	-
1	160	'blue'	$>$	$=$
2	100	'red'	$<$	$\neq$
3	120	'green'	$=$	$\neq$

Table 2.6.: Example for the deduction of critique patterns. In this e-commerce scenario every item is described by two features - price (numerical in $\mathbb{N}$) and color (categorical in {'blue', 'red', 'green'}). The current item is listed at the top and the critique pattern of each catalog item can be found in the same row.

The following outline of the working principle of MAUT-based dynamic critiquing is based on the proposition of the concept by Zhang et al. [56] and explanations in [47].
In multi-attribute utility theory (MAUT) it is assumed that the preferences of users can be modeled by the combination of utility functions for each feature of a multi-dimensional item space. Zhang et al. propose to compute an ad-hoc user profile representing the importance of the items' features and use this profile to determine the most preferred compound critiques. For the generation of the profile the weights representing the importance of the features are updated in each cycle. Every feature involved in a selected critique is increased in its weight. The set of critique options is then determined in a two step process: Firstly, the available items are ranked according to the deduced utility function using the aggregated weights. Then for the top items a compound critique is defined relative to the currently presented item, for which the derived filter would match the respective item. As this approaches implementation is not relevant for this work, please refer to [56] for an extensive description and formalization.

While Apriori-based dynamic critiquing fixes the central problems of static unit critiquing the problem of weak relevance of the proposed compound critiques becomes relevant. Tackling this problem is the aim of MAUT-based dynamic critiquing as the critique options are chosen based on the user's preferences. However, as shown by Reilly et al. [47] in their comparative study the efficiency (by number of cycles) of these systems is quite similar.

Incremental Critiquing

The idea of MAUT-based dynamic critiquing to construct a profile over multiple cycles instead of modeling the user solely by his current feedback underlies the approach of incremental critiquing, as well. Yet, a central difference between these approaches lies in the fact that in incremental critiquing the constructed profile is used to select the next item (that is, to involve it in the SelectNewItem function) while in MAUT-based dynamic critiquing the GenerateCritiqueOptions function is based on this profile. The following explanations are based on the paper by Reilly et al. [46].

Incremental critiquing is designed as an extension to Apriori-based dynamic critiquing but can, nevertheless, be applied for other critiquing approaches. Therefore it is identical to Apriori-based dynamic critiquing with respect to all algorithms except for the selection of the next item. A user profile is constructed and maintained as a set of past unit critiques, which are somewhat interpreted as one compound critique for constructing the filter of the current cycle. In each cycle the selected unit critique or the set of unit critiques equivalent to the selected compound critique are added to this set. To ensure consistent and non-redundant unit critiques in this profile a pruning mechanism is applied afterwards: Previous unit critiques referring to the same feature as the newly added ones are considered. Firstly, less restrictive critiques are removed. Less restrictive means that their inherent filter is implied by the filter of at least one of the newly added critiques. Secondly, contradicting critiques are dropped. Those are critiques inducing a filter that cannot be satisfied by any feature value at the same time as the filter of at least one newly added critique.

Finally, the generated profile is used to refine the next item selection process: Like in dynamic critiquing first the current critique is used as a filter to receive the catalog of possible items. This set is then ordered according to the product of similarity to the current item and compatibility to the past critiques. The compatibility is measured by the proportion of the satisfied critiques from the whole set of past critiques. Hereby items both most similar to the current one and compatible with the user profile are ranked higher. The first item in this ranking is presented to the user as the next item. This process is formalized in algorithm A.3.

The results of the comparison of Apriori-based dynamic critiquing and an incremental critiquing recommender system by Reilly et al. [46] indicate that incremental critiquing can reduce the length of recommendation sessions by up to 50% for two given example item domains. Hence, it can be said that incremental critiquing has a tendency to be more efficient than the previously presented approaches.

2.4.5 Extensions to Critiquing-based Recommender Systems

The basic critiquing approaches described above can be extended or altered in various ways. In this subsection we will focus on a small selection of those extensions which are relevant for comparing and motivating the implementation of this work. For a comprehensive overview on recent advances in the field of CBRS refer to [34] and [16].

We will first discuss alternative critiquing implementations differing significantly from the basic ones, afterwards introduce the concept of user-motivated critiques and shine a light on the notion of explanations in critiquing-based recommender systems.

Hybrid Approaches for Critiquing-based Recommendation

There exist several critiquing-based recommender systems going beyond the pure ad-hoc-user-profile- and knowledge-based recommendation of the basic approaches discussed in the previous subsection. These CBRS rely on hybridization designs involving other filtering techniques or user profile learning. Here we will give a brief overview on *tweaking critiquing* [35], *example critiquing* described in [16], *history-guided ConvRS* [50] and *user adaptive ConvRS* [26].

Tweaking Critiquing Tweaking critiquing is designed to prevent prolonged recommendation sessions in dynamic critiquing due to false-leads. Here in each cycle the top $k-1$ similar items to the last one and the last one itself are proposed, of which the user choses one to critique. This adds a comparison-based component to critiquing and enables to revert a previous critique, if it turned out to be a false-lead [35].

Example Critiquing Example critiquing adds the possibility for selecting relative values as critiques to the presented item (such as "Feature x should have a value y smaller than this!") instead of the usual directional and replacement ones found in the basic approaches. Additionally, all of those critiques may be combined to a compound critique hereby basically specifying an example desired item relative to the presented one [16].

History-guided ConvRS History-guided conversational recommendation extends incremental critiquing RS with unit critiques only. It relies on treating every critiquing path in a session as a problem in the sense of case-based reasoning. The history of past successful critiquing paths logged during the interaction of previous users is treated as the case-base. Now the most similar paths to the current one are selected and a set of potential next items is formed by choosing the final items of those paths. A next item meeting the past critiques is chosen from this set [50].

User Adaptive ConvRS The *Adaptive Place Advisor* (APA) by Göker et al. [26] implements what we call user adaptive conversational recommendation. A user profile is learned over multiple sessions. While offering advanced critiquing options to the user during a session this profile is updated differing between multiple types of preferences for the critiqued items. The learned user profile of previous sessions is used to predict a query used to determine an entry point for the dialog [26].

We will refer to the above systems for discussing the design space and possible approach for the implemented conversational recommender system of this work.

Extending Critique Options and Critiques

So-called *user-motivated* or *user-initiated critiques* can enhance both the user perceived comfort, as well as the recommendation performance through a reduction of session lengths [16]. CBRS involving this form of critiques differ from the basic approaches in the fact that they allow the user to specify (compound) critiques which are not contained in the set of proposed critique options. This solves the problem of weakly relevant feedback options 2.4.3 and has been applied in various conversational recommender systems [34].

Another important extension to critiques and critiquing options are explanations in CBRS. In critiquing-based recommender systems it is especially important to give reasons for the presented item and enhance the understanding of the item space as this lies at the core of the needs elicitation process. It is argued that dynamic critiquing inherently offers explanations on the item space, as the user may understand the trade-offs of the features in the domain [34]. However, there exist more sophisticated approaches relying on, for example, utility values of items according to MAUT, differences between the presented item and the given constraints or query, or based on the quantification of the level of confidence for a recommended item by the RS [31]. A complete outline can be found in [34] or [31].

3 The Conversational Recommender System for Business Flight Traveling

So far we have discussed the background on recommender systems and gave an overview on selected data mining and machine learning techniques. Having the introduced tools and guidelines in mind we will elaborate on the development and design of the conversational recommender system of this work in this chapter. In the following we will refer to this implemented RS as the *Business Flight Recommender System* (BFRS).

For a thorough analysis on the design space and a conclusive argumentation for and against certain approaches we will first give an introduction into the domain of business flight traveling and informally define the requirements for the RS using the parameter model introduced in subsection 2.2.4. Based on this descriptions we will afterwards discuss the basic recommendation approach considering different filtering techniques (described in 2.3 and 2.4). This will form the basis for the subsequent specification of the concrete design and architecture of the conversational RS in the context of alternative solution algorithms. This specification relies on the component model described in 2.2.5.

3.1 Business Flight Traveling Domain

For the development of valuable recommender systems it is important to have a clear understanding of the underlying application domains. Leaned onto the model of parameters for recommender systems described in subsection 2.2.4 we will give an overview on the users and items found in the domain of flight traveling or rather more specifically in the domain of electronic booking of flight journeys in a business context.

Firstly we will give an overview on the business flight traveling market and its customers' behavior, afterwards introduce terms and concepts of the underlying item domain and give a complete definition of the requirements for the Business Flight Recommender System.

3.1.1 Consumers and Products in the Business Flight Traveling Domain

In the scope of this work lies the application domain of *electronic booking platforms for business flight traveling*, which is a market sector of the more general business flight traveling market. In this work we will consider the German market sector of electronic booking platforms for business traveling in the first place. For the sake of simplicity we will, nevertheless, refer to it in the rest of this work by the more general name of business flight traveling.

In this subsection we will not go into the details of the underlying market itself, but only introduce necessary terms and concepts on the item domain of flight journeys, as well as highlight aspects of its consumers' behavior relevant to define the requirements for the BFRS.

Products in the Business Flight Traveling Domain

In the business flight traveling domain we call each product, as a whole, a *(flight) journey*. These products, however, have to be described by a hierarchic component architecture, where different domain-specific terms and concepts are used on each different layer. We will now elaborate the structure and nomenclature of this architecture starting at the most abstract level.
We generally differ between *one-way* and *round-trip journeys*: Every one-way journey describes a trip from one origin to one single destination, while a round-trip ones is defined as the two trips back and forth from an origin over the destination back to the same origin. The trip back to the origin departs after a certain usually chosen period of time. Every flight journey, independent of its type, is made up of at least one *(flight) itinerary*, which refers to the route or plan for a part of the journey. This means, that every one-way journey consists of one single itinerary, whereas each round-trip journey is described by an ordered sequence of exactly two itineraries. Such a route is a sequence of multiple *flights* or *flight connections* each representing the atomic connection between two airports. For an illustrative overview on these terms refer to figure 3.1.

With respect to the features of flight journeys we can again structure their description in the same hierarchy mentioned above. Starting on the most concrete layer, we usually describe flight connections by various properties, like for example duration, price, class of service, date of departure and number of stops. These features have in common that

they can be directly associated with one single real flight taking place at some point in time and are usually all given in a structured and explicit manner. Contrary to this, we can only define aggregated features on the more abstract levels of the architecture. This means, those features are derived from multiple underlying flight connections. On the level of itineraries we may describe the sequence of flights by accumulating their prices, durations or define an overall origin and destination. This applies for the level of journeys, as well.

On the first glance, hereby two distinct approaches for describing the products of the flight traveling domain exist: Firstly you could describe journeys by nothing but their aggregated, most abstract features. This is a usual perspective when comparing many journey options as the concrete routes and flights do not matter in such a scenario. The second approach is, of course, to describe journeys by the sequence of most basic features. This, however, is only useful if we compare a small set of journeys which are very similar in their aggregated features. Additionally, the aggregated features of journeys have an additional, informational value as the features of each flight in the sequence of the journey themselves lack the context of the whole journey. Yet the journey features are only determined by the underlying flight features and are not generated from external information sources; for example, there are no journey IDs as they are only composed items. Therefore, we argue that neither of the two approaches alone are useful for describing journeys, but rather both aggregated and flight-level features are necessary for a complete specification from a traveler's perspective.

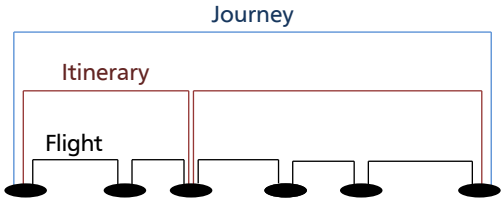


Figure 3.1.: The terms of the business flight domain arranged according to their hierarchic relation. Each journey consists of up to two itineraries, each itinerary is described by a sequence of flights. Each flight is characterized by elementary features derived from the real flight illustrated by the ovals as the arrival and departure locations.

Consumers in the Business Flight Traveling Domain

We understand the set of consumers on the market of business flight traveling to include employees of any company which travel regularly for the purpose of doing business. This is in accordance with the definition by the *German Business Travel Association* [1] in their report [5], to which we will refer for statistical assumptions on the market. According to [5] there were 11.3 million German business travelers in 2016 going on 183.4 million business trips. This means by the naïve combination an generalization of those numbers, that on average every business traveler goes on ca. 16 business trips per year. This number, however, includes all kinds of business trips regardless of the means of transport and should hence be considered with caution.

With respect to feature preferences on flights from the business travelers' perspective, the surveys in [5] indicate that properties like the opportunity to sleep or relax, to work and access the Internet during intercontinental journeys are indeed very relevant services to the travelers. While this only accounts for intercontinental flights this gives insight in the fact that the travelers themselves book products considering a variety of aspects apart from the purely economic ones like the price or duration.

3.1.2 Application Context of the Business Flight Recommender System

As explained in subsection 2.2.4 recommender systems' parameters are significantly determined by the software application they are integrated in and the expectations of the various stakeholders.

In this subsection we will focus on this aspect and introduce the application context of the BFRS first with respect to the surrounding software and afterwards consider the stakeholders' interests on the whole system. This will form the basis for an analysis of parameters and requirements to design a valuable RS for the given application domain.

[1] https://www.vdr-service.de/service/welcome/

Electronic Flight Booking Platform

The status-quo of the electronic flight booking platform is a pure information retrieval system by the definition given in 2.2.1: The users may start the retrieval process for finding a flight journey matching their needs by submitting a query. Each query consists of a vector of structured feature values describing the ideal flight from the user's perspective. The components are, for example, a concrete departure date, a preferred airline etc. These values have to be given by the user without any knowledge of the available items requiring him, like it is usual in IR systems, to posses deep domain knowledge.

The user query is afterwards passed to an external information retrieval engine searching for a subset of all flight journeys consisting of an arbitrary number of flights. This retrieval process and the underlying flight database, however, are completely transparent, as they are of high business value to the external service provider. We will refer to this system as the *flight retrieval system* or *engine* for the rest of the chapter. For each query a ranked list of up to 200 journeys is returned by the external service. This result set is filtered according to the rules and guidelines defined by the employer of the user. From this processed list the top ten to twenty items are presented to the user. The presentation consists of a graphical description of various journey-level features and can be extended to single flights features, if wished by the user. He then may select one product and proceed with booking or start a new query.

The recommender system is integrated into this platform as an additional processing step on the ranked list of candidates returned by the external provider's system. For a start, the filter based on the restrictions on journeys defined by the employer of the user should not be taken into account. This means, that the BFRS may consider all candidates and determine a subset of most interesting ones to the user. Additionally, the user interface towards the recommender system is supposed to support a chat-like interaction with the user. In the final software application an automated chat client (a so-called *chatbot*) will render the user interface and pass user feedbacks and requests in a structured manner towards the underlying recommender and retrieval system. Yet the aspects of natural language processing and dialog structures are of no concern to this work and we will abstract this user interface to a manner of passing structured messages. In the following we will only refer to this as the *chat layer* or *chat interface*. For a comprehensive overview on the application software system components and integration of the recommender system refer to figure 3.2.

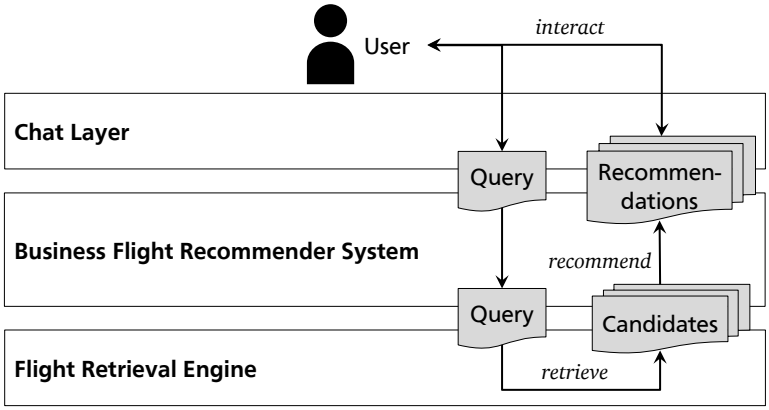


Figure 3.2.: The user depicted at the top interacts with the BFRS via the chat interface. The structured query is inferred by the chat layer and passed to the flight retrieval engine returning a set of candidates. From this set recommendations are generated under consideration of further structured user input.

Stakeholders and their Interests

Based on interviews with domain experts we determined three central stakeholders for the Business Flight Recommender System and their interests described in the following: The platform providers, the users (i.e. the business travelers) and the companies employing the travelers. As the developed, prototypical recommender system should ignore the rules and guidelines defined by the companies, we will only give a brief overview on the users' and provider's interests, while not going into the details on the party of employers.

The platform provider is in general interested in increasing his revenue, which is proportional to the number of bookings received via the electronic platform. Secondly, the resources necessary to employ the recommender system with respect to computational, bandwidth and storage requirements should tend to a low level, yet this aspect is definitely not of primary concern for the prototypical system. The user's have several equally significant interests: First of all, the time spent searching for a best or at least acceptable journey should be as low as possible. Secondly, the

resulting journey should meet their needs best. Lastly, there might be some users with certain expertise in the field of traveling due to the frequency of their business trips, who prefer to have more control over the search process, while many others most certainly posses few or no knowledge on the flight domain. The later ones especially desire to explore the journey options with the least cognitive effort while having the most certainty for their decision when booking.

Based on this abstract description of requirements and available data we will give a comprehensive definition of requirements in the subsequent subsection from an implementation oriented view.

3.1.3 Parameters for the Business Flight Recommender System

In this subsection we give a comprehensive list of parameters for the Business Flight Recommender System derived both from the previous domain description and the concepts introduced in the background section 2.2. We will especially refer to the model of parameters (in subsection 2.2.4), the model of components (in subsection 2.2.5) and the common, general problems in RS (in subsection 2.2.7). The following listing can be considered as the general analysis and description of requirements and restrictions for any RS in the domain of business flight traveling, as well.

For the sake of clarity we will split the parameters according to the dimensions of the introduced model: First we will list the parameter referring to the user model, to the data model and finally the application model. We will associate each with a set of requirements and restrictions useful for the design steps in the next sections.

Parameters and Requirements of the User Model Dimension

The parameters of the user model dimension basically define how the actual user behavior and desires are defined without taking into account their formal representation. Along with each parameter's characteristic a set of abstract requirements is implied.

Medium-sized, Weakly Dynamic User Base From the application of the RS in the electronic booking platform results a small to medium-sized user base as the access to this service is restricted by company contracts. The user base is dynamic, but significant changes only occur at rare points in time, where new companies are contracted.

Heterogeneous Domain Expertise While there are experts in the field of business flights, for example simply due to great experience through frequent traveling, there are many employees traveling seldom. This leads to a lack of expertise on their side. The RS should both enable experts and novices in the flight traveling domain to receive valuable recommendations. Additionally, for sparse feedback vectors, as they are given for the casual travelers, the recommendation quality should not deteriorate.

Timely Dynamic Preferences The timespan between the booking of two flight journeys by a casual user may be very long compared to for example movie streaming domains, due to the fact that flights require quite high economic effort. As the users' preferences usually change over time, adequate assumptions on their stability over these long time intervals between bookings have to be taken into account. This implies that a decay mechanism for old feedback in the user profile is required.

Fast Exploration of Options Both experts and novices of the user base require an overview on the available journey options to receive an adequate level of certainty for booking a journey. They will spent significant time on achieving this goal. For the casual users explanations of the item domain are required in addition. Therefore some form of exploration mechanism to discover the journey options is required.

Adaptive Recommendation For frequent fliers the notion of habits should be integrated into the BFRS. Users often have strong preferences for certain journeys or features of journeys, which should be integrated into the recommendation algorithm to enhance the speed and quality of receiving valuable propositions. In that sense, the gray sheep problem should be tackled, whereas the phenomenon of overspecialization is not critical, because users will have more trust in journeys alike the ones they have already booked.

Aim-oriented, Heterogeneous Desires in Traveling The travelers, by definition, have the desire to reach a destination region starting at a region of departure for a certain point in time. These requirements have to be met, to a certain degree, by any recommendation. That means, that parts of the user desires are session dependent and aim-oriented. In that manner, travelers are often willing to accept the best compromise for a given set of options although their general preferences differ. Therefore a certain heterogeneity of preferences over all sessions has to be dealt with; statistically speaking this can be perceived as a form of noise. Additionally, it can be assumed that the users are aware of their rough aim and can specify it explicitly with concrete values.

Travel Context For business travelers it is not unusual to book journeys spontaneously and with a short timely distance. Meaning that contextual information like the current location etc. can influence the current user's needs strongly. This can be taken into account for the BFRS.

Parameters and Requirements of the Data Model Dimension

The parameters of the data model dimension refer to both the actual item properties and their representation. The associated restrictions arise from the nature of items given by the domain while the requirements mostly relate to the mapping process of items and user feedback to explicit data structures.

Semi-Structured Features The flights are specified via a semi-structured format (concretely via an instance of the *Extensible Markup Language* (XML)). This means that the journeys are stored in a hierarchic tree structure, where the features of the flight connections are the leaves.

Flight Sequences and Feature Levels There exists a finite amount of objective features per flight, of which some are always given and some are optional information. However, each journey cannot be limited to a maximum number of underlying flight connections, as there is no knowledge on the flight retrieval system. The aim RS has to be able to deal with an arbitrary, finite and not necessarily equal number of flights specifying each journey. The aggregated features (refer to subsection 3.1.1) of the itineraries are partially given (e.g. total costs) or have to be deduced from the underlying flights. The overall journeys features are not explicitly stored. The RS has to be able to deal with the different levels of abstract feature descriptions.

Implicit Journey Features While the flights' most relevant properties are explicitly specified, there are several features which are not given and cannot be deduced from the given data without additional knowledge sources. For example, there exist safety and punctuality statistics on airlines and airports influencing the decision making process of travelers, which are not defined within the flight feature vectors. Therefore this can be viewed as an instance of the limited content-analysis problem and should be tackled by the BFRS - at least to a certain degree.

Highly Dynamic Item Catalog As the journeys are specified by combining multiple flight connections, there is a theoretically finite set of products, if the set of flights is finite. The underlying flight database can be understood by two perspectives: Every flight is identified by a *flight number* and every item instance having the same flight number, regardless of the point of departure, is treated as one. Whereas one can distinguish each flight connection by all its features; that is basically by the actual flights they represent. By the second perspective the item space is highly dynamic as many new items are added to the catalog every day. From the viewpoint of the first approach the catalog is not as dynamic, yet, due to the nature of the domain, many flight connections are canceled or changed over time. In both cases the aim RS definitely is required to prevent the new item problem.

Parameters and Requirements of the Application Model Dimension

The application model parameters are related to the user interface and the definition of usefulness of a RS inside its framing software application. Relevant requirements and restrictions mostly arise with respect to the feedback interface of the RS towards the user.

Conversational Interface As the Business Flight Recommender System is intended to be integrated inside an automated chat application the style of presenting and triggering recommendations (refer to 2.2.8) should be adequate for this scenario. First of all, this means that only a small number of items may be presented to the user at once, as this would introduce itself an overload problem. Secondly, ad-hoc user feedback on recommended items is possible and should be integrated in a dialog style.

High Timely Performance As the retrieval process of the flight retrieval engine itself is itself timely expensive, the recommendation process performed on the candidates should not introduce great overhead by high computational complexity. This, however, is only a soft guideline for the prototypical RS.

Efficient Recommendation Sessions The central quality of the recommender system should be a high efficiency with respect to the effort spent to receive a valuable recommendation by the RS. Here effort is defined in the broader sense of the number of trials to find the best item and the cognitive load imposed on the user.

All in all, after having carried out this parameter analysis or requirements elicitation we have received the basis for designing a valuable RS according to the guideline introduced by Picault et al. [41]. In the following section we will go into the details of designing a recommender system for the business flight domain, that meets the mentioned restrictions and requirements.

3.2 Basic Approach of the Conversational Recommender System

Based on the description of the application domain and the knowledge on strengths and weaknesses of filtering techniques for RSs we will give an overview on the implemented approach of the Business Flight Recommender System. For this purpose we will not only give a description of the architecture of the BFRS, but also reason for and against certain approaches. We will emphasize the strengths of our approach and highlight the novelties of the implemented RS. This overview will later serve as the basis to specify each component in detail, analyze its potential from an a-priori point of view and classify it according to the various taxonomies and terms introduced in the background sections.

In a first step we will interpret the given problem as an information filtering task and afterwards split it into multiple reasonable sub-problems. Then we will discuss the adequacy of the different filtering techniques and argue for a knowledge-based approach at the core of the RS we implemented.

3.2.1 Structuring of the Business Flight Recommendation as a Personalization Problem

In this subsection we focus on the brief analysis and interpretation of the given problem for finding the best flight journeys for every user as an instance of a problem solved by information filtering systems. This step basically renders the proof or indication of validity of the pursued approach of employing a recommender system and not using a pure information retrieval system like for example a search engine.

First of all, we will show that the given problem is an instance of an information overload problem in the context of persistent user tastes. We will give good reasons for a personalized selection mechanism, that is the use of a RS, and hereby classify the solution space of this problem to lie in the set of IF systems. Afterwards we will structure the underlying user preferences and analyze them with respect to item space

Information Overload in the Business Flight Traveling Domain

As described in subsection 3.1.3 there exist two groups of users, the professional ones and the casual ones, of which at least the latter type has no greater experience in the flight domain. That means those users can be said to be overwhelmed simply by the complexity of features and relations amongst the items. But even for the experienced business travelers several aspects of the item domain might introduce an information overload problem: First of all, the item catalog is highly dynamic making it difficult to have a constant overview on the relevant journey options for a business trip. Secondly, the features of the flights are of a high complexity which goes beyond structured vectors of properties. Therefore even for domain experts it can be difficult and costly (in terms of time) to weigh the pros and cons of possible journeys and their feature levels.

It becomes apparent by the definition of the information overload problem (refer to 2.2.1) that the given domain and task should be interpreted as such. Therefore the application of a recommender system as a form of an information filtering system could reduce the gravity of this problem and enhance the user experience. Now we will have a closer look at the underlying preferences to structure the problem according to the structure of the user behavior.

User Preferences in the Business Flight Traveling Domain

We argue that business travelers' desires and preferences can be described best within a short- and long-term model as it was already introduced in the motivation of contextual recommendation in subsection 2.2.9. In the domain of business journeys differences between the short-term preferences and the long-term preferences are induced by the underlying, pursued task: Travelers desire a connection between two geographical points within a certain timespan, which basically renders the underlying task. This strongly influences the decision on which flight to pick. For example, a journey from Frankfurt to New York City with high comfort is obviously of no use to a traveler desiring to arrive at Miami. The long-term preferences, however, are still relevant, but decrease in their subjective importance depending on the underlying task. In the example above such long-term preferences could be the desire for high comfort, which is ignored, if the current need to arrive at Miami cannot be realized. It is noteworthy, as well, that the long-term preferences are persistent over a longer period of time and the short-term (i.e. task-dependent) preferences persist at least over one recommendation session. Both aspects a requirement for personalization according to the definition in subsection 2.2.1.

Hence an adequate user preference model is described twofold: Long-term preferences deducible from user interactions in past sessions and a task or context influencing the user preferences for every recommendation session. That means, for the given domain only contextual recommendation is an adequate approach, which is why the chosen approach for the BFRS is of that kind. In the following subsections we will always differ between the two induced sub-problems of recommending in the given domain: Recommending based on long-term and short-term preferences.

Relation between User Preferences and Journeys in the Business Flight Traveling Domain

As business flight journeys are compound items consisting of sequences of flights on the lowest layer of abstraction, the user preferences can be structured accordingly. While for most application domains of RS, like for example movies or books, the preferences are related to atomic items and, depending on the approach, their features, for journeys the user can either perceive the product as a whole, split into the itineraries or most basically as a sequence of flights with their features.

From this different levels of item descriptions two ways of modeling the item space with respect to the user preferences are possible: Either the RS should determine the preference for each single flight and combine them to an estimation of user preferences for a complete journey, or alternatively consider each journey as a whole and derive recommendations from those compound items. This is basically the difference between setting the item space to be the set of journeys or the set of flights. We argue, that the first approach is not adequate, as the preferences for a single flight are always dependent on the other flights in the same journey. For example, a traveler may prefer the flight connection from Berlin to Frankfurt when departing at Berlin, but probably will not consider this flight valuable when departing at Munich, requiring him to take a long detour over Berlin. Therefore, no valid RS could be defined on the atomic flights, but only one that considers both abstract journey features and the concrete flights.

In the following subsections we will therefore only consider the recommendation problem by the item space defined as the space of journeys. This renders the basic assumption of the implemented BFRS, as well.

3.2.2 Contextual Recommendation Designs for the Business Flight Recommender System

In the previous subsection it was discussed that contextual recommendation should be chosen to adequately model the user preference given according to the parameter description in subsection 3.1.3. In this subsection we will elaborate on the possible contextual design of the system and give good reasons for the prototypical approach chosen for the Business Flight Recommender System.

For this purpose we will first define the concrete design options for contextualization in the given domain and afterwards discuss their adequacy for the problem.

Concrete Contextual Recommendation Designs

Contextualization of recommendations can be realized via three design approaches described in 2.2.9: Pre-filtering, post-filtering and contextual modeling. For each approach it is assumed that the utility function assigns each user and context pair a score for every item. For the given application domain this leads to the three concrete designs for contextual recommendation:

The first approach, in accordance to pre-filtering, extends the history of past bookings (with ratings) by the underlying task, for example represented by the triple consisting of departure location, arrival location, date and time of departure, and uses it as an input for a RS modeling users by their behavior in past sessions.

For a post-filtering design the history of bookings without task information is used as a basis for a RS modeling users by their past behavior. The result, in turn, is processed considering the dataset of past bookings together with their associated context and the current context. This second step can be carried out by RS both based on behavioral user models, as well as based on domain knowledge (e.g. using ad-hoc user profiles).

With respect to the contextual modeling approach as a concrete design, here every entry in the booking history of a user would be assigned the underlying task and treated as an additional item feature of the past journey. The current journey's features are extended by the current context's description. Here again a RS based on a behavioral user model could be applied.

Adequacy of the Contextual Recommendation Designs

We will now compare these concrete approaches with respect to their adequacy for the domain of business flight traveling. First of all, it becomes apparent that the pre-filtering technique can reduce the number of relevant ratings for a given task drastically. In the case of making a new journey, that is for example one with a new departure and arrival location, no previous bookings would be considered and the adaption mechanism would have very few samples. Independent of the concrete modeling of context when considering the average number of ca. 16 journeys by business travelers per year (see 3.1.1) the pre-filtering approach is not likely to grant good recommendation performance.

Regarding the contextual modeling approach we argue that this is an adequate design, albeit requires complex context modeling and recommendation techniques detecting feature dependencies to incorporate the underlying task in a useful manner. Additionally the downside of fitting the recommendation algorithm on the extended feature space is that abstract, context-independent feature preferences might be considered by a too low weight.

The post-filtering design compensates these shortcomings of the second approach: The initial RS recommending without

contextual considerations focuses on the context-independent preferences. Additionally here the contextual model can be adjusted in a later step and independent of the recommendation without context. This is a very valuable property for designing a prototype as a baseline which is the case for our problem.

All in all, it becomes apparent that both post-filtering and contextual modeling are adequate approaches to handle short- and long-term preferences in the business traveling domain. For the BFRS we decided to implement a post-filtering scheme, as this is more adequate for designing a prototype without additional data on user contexts. This allows the clear separation between long-term and short-term preferences and allows to split the recommendation problem for the Business Flight Recommender System into the two associated sub-problems. However, one should have in mind that this design decision implying this clear separation might not be appropriate in the end. Answering this question will be one goal of the evaluation of this system in section 4.

3.2.3 Adequacy of Filtering Techniques for the Business Flight Recommender System

Directly connected to the above considerations we will give an overview on the adequacy of the filtering techniques and their application in the described contextual architecture. Here we will not yet discuss the possible hybrid architectures for the BFRS, but remain on the abstract level of problems and approaches.

First we will analyze the adequacy of all introduced filtering techniques for the long-term preference recommender system based on the overview in the respective subsections in section 2.3 and the taxonomy introduced in 2.3.1. Afterwards we will carry out a similar analysis for the short-term preference-based recommendation

Adequacy of Filtering Techniques for Long-term Preference Recommendation

After reasoning for a post-filtering approach for contextual recommendation we will now focus on the recommender system considering the long-term preferences. According to the definition of IF the user preferences have to be deducible from user behavior 2.2.1, which for the long-term case manifests in the feedbacks on journeys across previous sessions. Therefore only recommender systems modeling the user by his past behavior, that is content-based RS or rather knowledge-based RS with user-profile-learning and collaborative filtering, are of relevance for solving this partial problem. We will now give a brief discussion of their adequacy for the given task of recommending flight journeys based on past preferences.

From the discussion of parameters of the business flight traveling domain most significantly three factors render collaborative filtering a probably less performing approach than knowledge-based RS for the given task. Firstly, the long-term preferences are timely dynamic, which means that ratings from a long time ago do not necessarily apply for today any more. Therefore either the user-item-matrix at the core of CBF is reduced to contain only new data (implying higher sparsity and often lower performance) or the recommendations are obsolete. Due to the heterogeneous user base, the recommendation quality for frequent fliers is reduced even stronger, because the casual users basically add no feedback information to the community. Secondly, the item catalog cannot be straight forwardly mapped onto a set of item IDs required for the CBF approach. Because flight journeys do not have domain-given unique identifiers, a feature-based assignment would be required, as for example a journey from Frankfurt to New York City on the next Monday would need to be considered different from one leaving on Tuesday. This, however, implies that a pure collaborative filtering approach neglecting all item features is simply not possible. Thirdly, a hybrid RS based on CBF is possible, yet would most likely under-perform, because the item catalog (even mapped to IDs by the item features) is relatively dynamic. Here the fact that CBF RS suffer from the new item problem becomes relevant. With respect to user comfort collaborative filtering is inherently less suitable, due to the fact that the recommendations cannot be motivated or offer insight in the alternative journey options, which is a necessary property for the given application.
We draw the conclusion that CBF RS are not adequate due to various parameters of the application domain.

Knowledge-based recommendation employing user-profile-learning has several advantages with respect to the above mentioned short-comings of CBF: First of all, the nature of flight journeys as compounds of multiple flights, each described by feature vectors, can be naturally included into KBR in the form of mapping the sequences of flights per itinerary to vectors of more abstract features. As indicated in subsection 3.1.1 the resulting abstract description of journeys is at least suitable for selecting a subset of interesting flights from all candidates. Secondly, the recommendations to the user can be reasoned in a human-understandable way by the RS, as they are deduced from the features of previous flights. Additionally, there is no requirement for coverage of the feature space or recommending new products, which is why the overspecialization on each user is generally not problematic. Finally, the high dynamics of the underlying

item set are naturally no problem to KBR and despite of the heterogeneity of the user base with casual users having a very sparse feedback history a knowledge-based approach can increase in its recommendation quality for the frequent fliers independently of the other users. Of course, the KBR approach implies several shortcomings starting at the sparsity problem in its general and specialized form as the new user problem. Additionally, there exist implicit features of flights and journey which cannot be deduced from the given ones that may influence the preferences of the users. Here KBR generally perform worse than e.g. CBF. Finally, the problem of obsolete recommendations due to old feedback has to be dealt with, as well, which can itself reduce the recommendation performance.

We draw the final conclusion that knowledge-based recommendation is the more adequate approach for the problem of adapting to long-term preferences and hereby generating recommendations. Nevertheless, the weaknesses of KBR have to be considered during the further design steps and during the evaluation.

Adequacy of Filtering Techniques for Short-term Preference Recommendation

For the short-term-preference-based post-processor theoretically any system, either modeling user's behavioral or non-behavioral, is an option, because the results of the long-term preference recommender system can be processed both based on only the current context or by considering the history of item ratings together with their context.
However, as the intended post-processor should only consider the short-term preferences of the current session, it should be realized by an session-specific, i.e. ad-hoc user profile under consideration of context. As this system relies on the determination of preferences, as well, we argue that it is a recommender system. Here only an ad-hoc-user-profile-based KBR approach is an option.

In the following section we will concretize the design of the Business Flight Recommender System split into the two basic recommendation components, long- and short-term-based, while the first is an adaptive knowledge-based RS, the second one is an ad-hoc-user-profile-based RS. Their combination and implementation introduce themselves more design questions to consider with the background of the parameter description of the BFRS.

3.3 Design and Architecture of the Conversational Recommender System

In the previous section we have discussed the basic approach for recommendation in the domain of business flight traveling. It was shown that a division of the overall problem into the recommendation oriented towards long-term and short-term preference is an adequate underlying model for designing a valuable recommender system in this domain. We argued for employing a user-profile-learning-based KBR for realizing long-term preference recommendation and ad-hoc-user-profile-based recommendation for post-processing the results of the first module. In this section we will elaborate on a concrete design for solving the twofold problem using the adequate recommendation techniques.

As a first step we will model the introduced problem formally starting at the underlying item, user and context model. This will serve as a basis to define the functional components of the implemented BFRS according to the model introduced in subsection 2.2.5 on an abstract and formal level. Finally, we will give a very brief overview on the concrete implementation with respect to the employed technologies.

3.3.1 Item and User Model of the Business Flight Recommender System

As described in the previous section we follow a knowledge-based recommendation approach for the design of the Business Flight Recommender System. Therefore, by the taxonomy of filtering techniques, the item model is defined based on domain-specific features and the user model may be described both in terms of past behavior and domain-knowledge together with ad-hoc profiles acquired from knowledge sources which go beyond ratings for items. In this section we will define the components associated with the user and item model according to the model in 2.2.5.

In this section we will first define a domain-based feature model of items, the item space model and the construction algorithm for the item profiles. Afterwards we will shine a light on the design options for modeling the user, constructing the profile and relating it to the model of the user space.

Item Modeling in the Business Flight Recommender System

In subsection 3.1.3 we gave a comprehensive overview on the parameters of the data model dimension, which is the basis for designing the models for each item and the item space. Every item in the actual domain is a composition of finite sequences of flights each described by structured features. Therefore we start at defining the space of flights as a model for the atomic flights and derive a *theoretical* and *practical item space* from it.

Every flight is given by a set of structured features of which several are only optional features and others are mandatory features. In the actual application this description is given in XML-format, but can be mapped onto a finite feature vector, where for each optional component an empty value is allowed. We call this value ϵ in the following. Therefore the space of flights $\mathscr{F}$ can be formally modeled as a finite space of features in the spirit of the VSM.

Definition 3.1 (Space of Flights) *We call $\mathscr{F}$ the space of flights or flight connections. Let ϵ denote the invalid or empty value for an attribute with the intended semantics of indicating the absence of information on that feature. Then $\mathscr{F}$ can be specified by an equation of the following form:*

$$\mathscr{F} = A_1 \times ... \times A_n, \text{ where } n \in \mathbb{N} \wedge \forall i \in \{1, ..., n\} : A_i \text{ is a finite set of feature values} \wedge \epsilon \in A_i$$

$\mathscr{F}$ hereby is a finite set of feature vectors of identical size.

Refer to table A.1 for a complete, informal definition of the values and their semantics of the feature domains $A_1, ..., A_n$ mentioned in the definition. In the very essence, the flights are specified by 18 features, of which 9 are mandatory and the other 9 are optional. However, there exist 3 complex features from the set of optional ones described by lists and maps of values which are modeled as (binary) attribute groups indicating the presence of a value or value of a mapping in this feature's list or map. This is possible as all values which can be assumed are known.

From the set of flights we can derive the set of itineraries defined as all finite sequences of arbitrary length, where every member originates from the set of flights. Here we make the important distinction between the theoretical and the practical space of itineraries: As the flight retrieval system responding to user queries is by its nature a black box, one cannot limit the length of itineraries or the possible combinations of flights without losing the claim to adequacy and completeness. Yet it is obvious that itineraries with ten underlying flights are very unlikely, if not even impossible or that a flight arriving in Berlin will probably not be followed by a flight departing in Miami in the sequence of flights. Hence we treat the theoretical itinerary space as the basis for the practical item space, which is only specified by being a real and finite subset of the theoretical set of itineraries. Therefore we receive the following definition.

Definition 3.2 (Theoretical and Practical Itinerary Space) *Let $\mathscr{F}$ denote the flight space then the theoretical space of itineraries $\mathscr{Y}_T$ is given by the set of all finite sequences upon $\mathscr{F}$, that means as:*

$$\mathscr{Y}_T = \{(f_k)_{k \in \mathbb{N}} | \forall k \in \mathbb{N} : f_k \in \mathscr{F}\}$$

Whereas the practical itinerary space $\mathscr{Y}_P = \mathscr{Y}$ is defined as a real and finite subset of elements from the theoretical itinerary space. As the concrete selection mechanism is unknown, the concrete subset definition is left underspecified, hereby modeling the black box aspect of the underlying flight retrieval system:

$$\mathscr{Y}_P = \mathscr{Y} \subset \mathscr{Y}_T \wedge \mathscr{Y} \text{ is finite}$$

For the final model of the item space we again distinguish between the theoretical and the practical item space. On the most basic level of describing the items each journey is directly associated with up to two itineraries. However, depending on the component of the recommendation algorithm, which we will elaborate in subsection 3.3.3, a vector of abstract features can be deduced to represent the items. As a consequence we define the set of items by the basic level of describing them via up to two itineraries and introduce concrete, deduced features for each component, where it is necessary.

Definition 3.3 (Theoretical and Practical Item Space) *Let $\mathscr{Y}_T$ denote the theoretical and $\mathscr{Y}$ the practical itinerary space. We then define the theoretical item or journey space $\mathscr{I}_T$ as follows:*

$$\mathscr{I}_T = \{(i_1, i_2) | i_1, i_2 \in \mathscr{Y}_T\} \cup \{i | i \in \mathscr{Y}_T\}$$

Analogously the practical item or journey space $\mathscr{I}$ upon the practical itinerary space:

$$\mathscr{I} = \{(i_1, i_2) | i_1, i_2 \in \mathscr{Y}\} \cup \{i | i \in \mathscr{Y}\}$$

During the rest of this work we will only consider the practical item space, because only for this *finite* subset of items the recommendation problem is well-defined.
We do not make further assumptions on the item set's structure, but only the ones mentioned in the data model parameters described in subsection 3.1.3. Therefore the item space model is identical to the space of item models. The item model construction algorithm for the basic case, specified above, is trivial as the domain given data is directly mapped to feature vectors, while the complex features (lists and maps) are resolved as multiple (binary) components.

User Modeling in the Business Flight Recommender System

The users are modeled according to the twofold of preferences: On one hand, a session-specific profile has to be created. On the other hand, a long-term profile based on past feedback needs to be maintained. As the Business Flight Recommender System is realized as a knowledge-based approach the concrete user models, however, depend strongly on the employed recommendation algorithms, which we will define in subsection 3.3.3. For now we simply give a underspecified definition of the user model split into an ad-hoc and a behavioral part represented by structured features.

Definition 3.4 (User Space) *The user space $\mathcal{U}$ is the set of user models, where every user is specified by a pair of profiles. The first component is element of the finite space of long-term preference profiles $\mathcal{U}_{long}$ and the second component is element of the finite space of ad-hoc profiles $\mathcal{U}_{short}$. Hereby the user space can be specified in the following form:*

$$\mathcal{U} = \mathcal{U}_{long} \times \mathcal{U}_{short}$$

The restrictions on quantity and quality of knowledge on user behavior described in 3.1.3 render the underlying user space model. The user model construction algorithm depends on the respective recommendation components and will be elaborated in the following subsections.

3.3.2 Feedback and Context Model of the Business Flight Recommender System

The feedback model of recommender systems describes the forms of user input, which are considered during recommendation (refer to subsection 2.2.6). Whereas the context model of a RS specifies. which context the user can be in, how it can be measured and finally how it can be modeled in a structured manner.

In this subsection we will first define an adequate feedback model differentiating again between user inputs related to a current session and those related to an overall behavior. Finally, there will be a specification of a context model and a definition of how to acquire it.

Feedback Modeling in the Business Flight Recommender System

As the users' preferences are divided into two groups, it is reasonable to differ between user input referring to long-term and short-term preferences, as well. We will now elaborate on these two forms of feedback.
Several psychological aspects and parameters depending on user expectations can be taken into account for choosing a valid feedback model (as described in 3.1.3 and 2.2.6). From a practical point of view, disregarding these considerations, the underlying dataset for developing valid system parameters is a central aspect, if the system should offer proper recommendation quality from the very beginning; i.e. without a possible on-line analysis. For the given underlying dataset of this work, as we will elaborate in section 4, only bookings are defined and no additional information on the preferences of users. For realizing a valuable evaluation on this dataset the decision on implicit feedback with a binary rating scale is a reasonable choice. These binary ratings can be acquired by recording which items where viewed and which of those were booked by the user. Here a booking indicates a positive rating ("like") and not-booking a viewed item may indicate a negative rating ("dislike"). This is an adequate approach for the feedback regarding long-term preference recommendation as it both meets the restrictions of the evaluation dataset and imposes the lowest cognitive effort on the user. Yet it introduces certain feedback noise, as the inference mechanism for negative ratings might falsely interpret reviewed, non-booked items, which are similar to the booked one, as negative without considering degrees of appeal to the user.

With respect to feedback specifying session-dependent preferences, usually there are employed explicit approaches as found in ad-hoc-profile-based, knowledge-based approaches. Here the user may specify a query, a set of feature values or other forms of preference definition. As the intended approach for the session-wise recommendation is a knowledge-based algorithm this modeling of short-term-related feedback is adequate. We will elaborate on the concrete feedback model for the respective module in the following subsections.

Explicit Context Modeling in the Business Flight Recommender System

As for the prototypical nature of the Business Flight Recommender System the user's context can be reduced to the context induced by the underlying task of each session, as described in the subsections 3.1.3 and 3.2.3. The abstraction from the environmental context of the current user, like for example his current geographical location or the current time of day, make the developed system a valuable baseline and allow future extensions (as discussed in 5).

The acquisition of the user context in the Business Flight Recommender System is realized in an explicit, in-transparent manner. This means, that every user starts a recommendation session by specifying a query. Hereby the BFRS can

be classified as an reactive recommender system (refer to 2.2.8). Each query consists of a sequence of a maximum of two sub-queries each referring to an itinerary of their aim journey. Every sub-query is given by the structured features specified in table A.2. In the very essence, every sub-query consists of 4 mandatory features extended by 9 optional features. These features are the domain-specific input fields required by the underlying flight retrieval engine. Therefore we receive the following formal model of context.

Definition 3.5 *Let $\mathcal{Q} = \mathcal{Q}_{sub} \times \mathcal{Q}_{sub}$ denote the space of queries, where $\mathcal{Q}_{sub} = Q_1 \times ... \times Q_l$ the set of finite, structured sub-query vectors. Then the user context space $\mathcal{C}$ can be specified as being identical to the set of queries: $\mathcal{C} = \mathcal{Q}$.*

We argue that this context model is adequate for the given problem, as the specification of the mandatory fields of a query requires only weak cognitive effort from the users and completely reflects their current task. Nevertheless, the specification by the users themselves can introduce errors, due to the fact that they have to make their needs explicit. Additionally, it is noteworthy that this perspective on a user-defined query as his context is quite unusual. However, in the domain-implied aim-orientation of travelers this perspective is indeed valid.

3.3.3 Hybrid Design of the Recommendation Algorithm of the Business Flight Recommender System

In the previous subsections the basis for defining the recommendation algorithm of the Business Flight Recommender System was given by discussing various design options for the components of an RS. In this section we will specify the recommendation algorithm on an abstract layer by introducing an hybrid architecture of the intended system. In the following sections we will elaborate on the implementation and design of the modules underlying this architecture.

For a comprehensive specification of the architecture we will first deduce the overall structure and interfaces of the underlying components by referring to the taxonomy in 2.3.6 and the observations from previous subsections. Afterwards we will give an overview on the concrete recommendation algorithms involved and their intended functionality as a black-box. The resulting architecture is depicted in figure 3.3 as an overview.

Cascading Pipelined Hybrid Architecture of the Business Flight Recommender System

To transfer the two-step contextual recommendation concept described beforehand, of course, a pipelined hybrid design is the best approach. Here, as the recommendations generated by the long-term-oriented RS should be refined, the cascade approach is fitting. Therefore the hybrid Business Flight Recommender System is divided into two pipeline stages, where the first component realizes an initial item ranking based on long-term user preferences. The second component refines the resulting recommendation list by an ad-hoc profile. We will refer to the first component as the *offsetting module* and the second one as the *conversational module*. These names will become apparent in the following paragraphs.

For the second RS in question either constraint- or case-based recommendation are potential designs for realizing the knowledge-based post-processing task. Due to the restriction implied by the low expertise of casual users of the system, a case-based approach is provably more adequate, as no concrete feature values have to be specified by the user (refer to 2.3.4). This aspect receives even more weight, when considering that the long-term recommendation is likely to have a low performance for casual users with few ratings. This has to be compensated by a RS suitable for elicitation of user preferences in an ad-hoc manner.
Here a conversational approach for recommendation is very promising for the given application as the heterogeneous user knowledge is accounted for by enabling fast navigation through the item options. In addition to that, the interactive dialog scheme of ConvRS perfectly fits the integration into a chat bot application. Therefore the second component of the pipeline, which we refer to as the *conversational module*, is realized as a ConvRS.

Parallel Hybrid Architecture of the Offsetting Recommender

In basic conversational approaches, like the one implemented for the RS of this work, the entry point is determined via a component finding the item best fitting to a given user query (refer to 2.4.4). This offsetting mechanism of the initial item itself can be viewed as a first component of a cascading hybrid RS. Therefore an obvious design option is to integrate the query-based mechanism into the first pipeline step of the BFRS - i.e. the component of the long-term recommender system. Hereby the first module can be viewed as an mechanism to offset the entry point for the conversational recommender system to get closer to the most desired item. As a consequence the dialog length of each session should be significantly reduced considering both long-term preferences and short-term preferences in the form of a query. This idea coined as *entry point decision making* [34] or entry point offsetting. In that sense we will refer to the first pipeline step as the *offsetting module* from now on.
To combine the query-based recommendation system into the first stage a hybrid design was chosen, as well. For a clear separation of long- and short-term preferences implied by the query, we propose to design the offsetting module as a parallel HRS, where the scores or ranks of each of the components are combined.

All this observations and design choices lead to the hybrid architecture depicted in figure 3.3. Here the offsetting module receives the set of candidates retrieved by the flight retrieval engine based on a given user query. The conversational module refines the recommendation of the first step in an interactive dialog. In the following subsections we will elaborate on each component, its underlying models and formal problem, as well as the implemented solution approach.

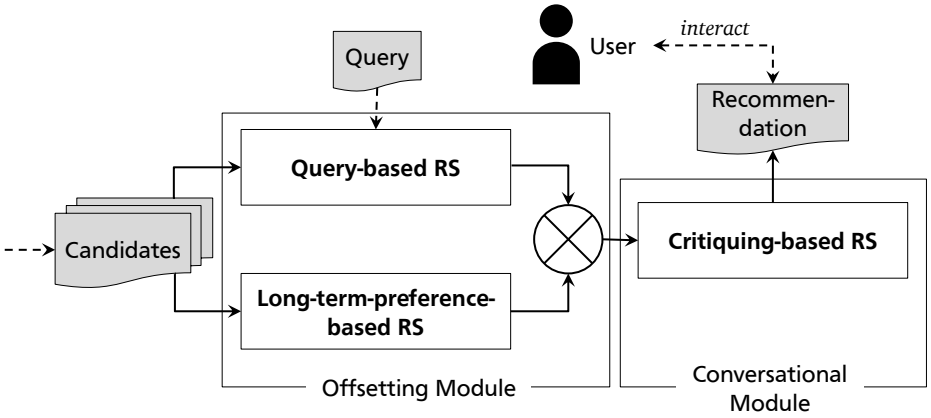


Figure 3.3.: The architecture of the BFRS depicted as a data flow graph. The offsetting module receives the input candidates processing them in a parallel manner and combining the results (represented by the crossed circle). The resulting ranked list of candidates is handed to the conversational module where the implementing critiquing-based RS generates recommendations interactively.

3.3.4 The Query-based Recommendation Component

In the previous subsection we introduced the overall architecture of the BFRS. The query-based recommendation component located in the first pipeline step, which we call the offsetting module, is intended to determine a ranking for the input items meeting the session-dependent user needs best. For the purpose of a complete formal and implementation-oriented description we will focus on this component in this subsection.

In a first step we will define the underlying formal problem of the query-based RS, describe the working principle of the implemented variant and give good reasons for the design decisions.

Formal Problem Specification

The query-based recommendation component receives the set of flight journeys retrieved by the underlying engine. On this set of candidates a ranking has to be defined under consideration of a given user query. The intended functionality of the module is to define a ranking which reflects the degree of congruence of the user query with each journey. Hereby the best item with respect to the current context defined by the query is determined.
To formalize this functional requirements as a problem, the context space $\mathscr{C}$ of user queries and the underlying item space $\mathscr{I}$ have to be considered. We define the following problem.

Definition 3.6 (Formal Problem of the Query-based RS) *The formal problem underlying the query-based RS is defined by the following parameters:*
Given: *A user query $q \in \mathscr{C}$ rendering the current session context and a set of candidate items $I \subseteq \mathscr{I}$.*
Wanted: *A utility function $f : \mathscr{C} \times I \to [0,1] \subset \mathbb{R}$.*

The basic idea for this module is to define a function mapping a given query and item to a degree of usefulness of the item with respect to the query.

Implemented Solution Approach

For the solution of the above problem we defined a function leaned onto the concept of hamming distance similarity measure (refer to 2.1.1). However, as the function is defined on two different input domains, it technically is no similarity measure. The given query is split up into the sequence of sub-queries each described by the features listed in table A.2. For determining the degree of utility of a journey with respect to a whole query, the journey is split into the sequence of itineraries which are each compared to the respective sub-queries. This accounts for the semantics of sub-queries to refer to one itinerary of the journey.

Now the degree of utility is determined by forming a weighted sum on the degrees of utility of each itinerary towards the respective sub-query. The utility function on the set of pairs of sub-queries and itineraries is itself realized by a scheme leaned onto weighted hamming similarity measure, where each feature of the sub-query is compared to the referred features of the itinerary. For this purpose the basic description of itineraries as sequences of flights is reduced to a feature vector of fixed size over corresponding features. Hence we can specify the implemented utility function on the space of contexts and candidates by the following definition.

Definition 3.7 *Let $I \subseteq \mathscr{I}$ denote the candidate set, where each candidate can be represented by a pair of itineraries represented by reduced, structured features, so that $i = (p,q) = ((p_1,...,p_l),(q_1,...,q_l)) \in I$ for $n \in \mathbb{N}$. For the space of contexts given as the space of queries with the same dimensions as the reduced journey representation, it holds that $c = (v,w) = ((v_1,...,v_l),(w_1,...,w_l)) \in \mathscr{C}$. Then the implemented utility function is defined as follows:*

$$f((v,w),(p,q)) := \alpha_1 \cdot \sum_{j=1}^{l} \omega_j \cdot g_j(v_j,p_j) + \alpha_2 \cdot \sum_{j=1}^{l} \omega_j \cdot g_j(w_j,q_j)$$

Where α_1, α_2 and for $i \in \{1,...l\} : \omega_i \in [0,1] \subseteq \mathbb{R}$ having $\alpha_1 + \alpha_2 = 1$ and $\sum_{i=1}^{l} \omega_i = 1$. The functions g_j for $j \in \{1,...,l\}$ are the utility functions for the respective features.

The weights α_1, α_2 and ω_j, as well as the functions g_j are adjusted and determined during the development phase of the evaluation 4. It is noteworthy, that the empty vector (i.e.vector of all invalid values) is added in the concrete implementation. This is necessary, because the journeys do not necessarily have the same number of itineraries as the number of given sub-queries. For those vectors every feature comparison returns a utility value of zero. This is the case, if the user queries for a one-way journey and receives round-trip journeys, or vice versa.

The described, implemented approach relying on a scheme similar to weighted hamming distances was chosen, as the features of sub-queries and itineraries have to be expressed with both numerical and nominal value domains. By adjusting the weights and underlying utility measures for each dimension the system can be easily adjusted.

3.3.5 The Long-term-preference-based Recommendation Component

Similar to the preceding explanations we will focus on one of the components of the hybrid architecture of the BFRS. In this subsection we will treat the long-term preference recommender system relying on a history of bookings as implicit ratings. This adaptive mechanism resides in the offsetting module.

In a first step we will define the underlying formal problem of the long-term-preference-based RS, describe the working principle of the implemented variant and give good reasons for the design decision made.

Formal Problem Specification

The long-term preference RS receives the set of candidate journeys as an input. The module should create a ranked list from the candidates, which reflects the degree of relevance to the current user according to his feedback history. The feedback history is, as specified in subsection 3.3.2, given by a timely ordered sequence of bookings considered as positive together with the previous candidates which were not selected in that session considered as negative.
To formalize the problem of long-term-preference-based recommendation an estimation of the utility function on the user and item space is necessary. The user model at least includes the aforementioned feedback history.

Definition 3.8 (Formal Problem of Long-term Preference Recommendation) *Let $I \subseteq \mathscr{I}$ denote the candidate set, $\mathscr{U}$ the set of users, where the space of long-term user preference models $\mathscr{U}_{long}$ can be specified by the following equation.*

$$\mathscr{U}_{long} = \{b = (b_i)_{0<i<=n} \mid n \in \mathbb{N} \wedge \text{ all members of } b \text{ are elements in } \mathbb{B} = \{0,1\}\}$$

Then the problem of long-term preference recommendation is defined as follows.
Given: *A user $u = (u_l, u_s) \in \mathscr{U}$ with $u_l \in \mathscr{U}_{long}$ and a set of candidate items $I \subseteq \mathscr{I}$.*
Wanted: *A utility function $f : \mathscr{U}_{long} \times I \rightarrow [0,1] \subseteq \mathbb{R}$.*

The basic idea of this module is to define an adaptive algorithm learning from the series of timely ordered and binary labeled samples. This algorithm specializes for every user and should be able to predict a label or degree of relevance for any item of the item space regardless of the underlying query or session.

Implemented Solution Approach

For the solution of the above problem the various probabilistic and similarity-based algorithms described for CBF, which can be transferred to KBR, in subsection 2.3.3 can be considered. For the implementation of the Business Flight Recommender System a Naïve Bayes classifier was chosen for multiple reasons: As described in 2.1.3 the Naïve Bayes classifier with smoothing offers a high robustness, which is necessary for the intended modeling. In the business flight traveling domain there may exist several outliers in the booking history of a user with respect to the actual preference model. This is caused by the fact that travelers are willing to agree on compromises on certain features, if in the set of available journeys there is no journey meeting their preferences for their underlying, given task. Additionally, during the evaluation the Naïve Bayes approach turned out to offer a solid baseline. It performed better than a reasonable k-Nearest-Neighbor approach implemented as a reference during the performance evaluation in section 4.

The implemented Naïve Bayes classifier solves a two-class-classification problem, where the labels are "relevant" and "irrelevant" (or 1 and 0). The classifier is trained on the past samples of a user; that means, his history of bookings and candidates. The utility score of each item is determined by applying the classifier and setting the score to zero for every irrelevant item and for every relevant item to the level of confidence for belonging to the relevant class. This level of confidence is determined by the probability of the Naïve Bayes classifier decision rule (see 2.9). Additionally, smoothing is employed to increase robustness. This is hereby a complete description of the construction of the utility function for solving the above problem.
The basic item space $\mathscr{I}$ is mapped to a feature vector space of dimensionality 9. Here the journeys are represented by the most abstract, aggregated features (such as total journey duration or price) adequate for the task of *long-term* preference prediction and offsetting the entry point from an abstract point of view. Various feature constellations were tested during the development phase, which emphasize different aspects, e.g. the time of the journey or comfort properties (see 4).

The hyper-parameters to adjust are, in the first place, the choice of abstract features and the discretion of numerical attributes. Additionally, the class of negative samples (non-bookings) are overrepresented, due to the fact that each booking is an item taken from a set of candidates with up to 200 members. Hence a mechanism for over- or under-sampling on the training data has to be chosen. In the final system configuration a majority class under-sampling approach was implemented (refer to 4 and 2.1.3). Finally, a mechanism for discarding outdated feedback was introduced by selecting a subsequence of the last entries of the booking history. For this purpose the number of most recent bookings taken in consideration for learning can be adjusted.

3.3.6 Hybrid Composition of the Offsetting Recommendation Component

In the hybrid architecture of the Business Flight Recommender System described in subsection 3.3.3 the offsetting module is responsible for finding an entry point for the conversational recommender system, which leads to the shortest dialogs in the conversational module. As previously described the query-based and the long-term-preference-based RS generate a utility function or ranking on the set of candidates. For determining the best entry point considering both components' results a compositional mechanism is required. In this section we will focus on this part of the RS.

Firstly we will define the underlying formal problem of the offsetting RS, describe the working principle of the implemented variant and give good reasons for the design decisions made.

Formal Problem Specification

As described in 3.3.3, the offsetting module is designed as a parallel hybrid RS to enable both a clear separation of the influence of the user adaptive mechanism and the query-based component, as well as a personalized ranking of items according to long- *and* short-term preferences. We define the formal problem of the composition of the results of the underlying two RS in a general fashion, which allows all parallel HRS formalized in 2.3.6.

Definition 3.9 (Composition Problem of the Offsetting Module) *Let u_1 denote the utility function estimated by the first RS and u_2 the one by the second RS. Further on let R_1 and R_2 be the respective associated ratings of candidate items $I \subseteq \mathscr{I}$. Then the composition problem of the offsetting module is defined as follows.*
Given: *The utility functions u_1, u_2 and their associated ratings R_1, R_2 on the item set I.*
Wanted: *A utility function u_{res} and an associated rating R_{res} combining the given scores and ratings.*

The basic idea of this module is to implement either a weighted, switching or mixed combination scheme from the results of the underlying recommender systems. Here the result should reflect the importance of long- and short-term preferences so that the resulting best item is the one actually most useful to the user.

Implemented Solution Approach

For the prototype of the BFRS a form of a mixed, parallel hybrid scheme was implemented. The resulting utility function is leaned onto the respective formula described in subsection 2.3.6. More concretely, the implemented algorithm determines the utility function by a two step process: First each item is assigned a score according to the long-term preference RS. Afterwards, for every item with a score below some threshold τ after the first step, the score according to the query-based RS is assigned. Therefore the utility function u_{res} is determined from the utility functions $u_1 = u_{long}$ and $u_2 = u_{short}$ of the underlying RS by the following equation, where each score is normalized by the normalization factor n_{long} or n_{short}:

$$u_{res}(x,y) = \begin{cases} n_{long} \cdot u_{long}(x,y) & \text{, if } u_{long}(x,y) > \tau \\ n_{short} \cdot u_{short}(x,y) & \text{, else} \end{cases}$$

During the evaluation it became apparent that the long-term preference RS offered a significantly better performance than the query-based recommender system (refer to section 4). By the above scheme both a form of a backup system and a blending of components' results is realized. The backup aspect resembles a switching, parallel HRS with a criterion connected to the degree of certainty of the first RS. If the long-term preference RS, which assigns the scores according to the probability of belonging to the relevance class (see 3.3.5), defines scores reflecting low certainty of the classification with respect to the threshold, the query-based RS replaces the scores by potentially more reliable ones.

The hyper-parameters to tune during the development phase of the evaluation are given by the threshold τ, and the normalization factors n_{short} and n_{long}.

3.3.7 The Conversational Recommendation Component

In the previous subsections we discussed the design of the offsetting module to determine an entry point for the conversational recommender system. Here we will focus on this second module and elaborate on its implementation.

Firstly we will define the underlying formal problem of the conversational RS, describe the working principle of the implemented variant and give good reasons for the design decisions made.

Formal Problem Specification

For the conversational module in principle all approaches discussed in section 2.4 have to be considered. For the restrictions on limited domain knowledge of casual users and the desire of all users to explore the space of potential journeys a navigation-by-proposing approach is inherently more adequate. Here the users have the chance to elicit their needs and get an insight in the item space efficiently. Therefore we will not further consider navigation-by-asking RS for the problem specification and solution definition. Additionally, the special form of critiquing-based ConvRS is the best option due to its best trade-off between user effort and low ambiguity of feedback (as discussed in 2.4.1).
The formalization of the underlying problem of the conversational module as a critiquing RS is directly derived from the algorithm of critiquing-based recommender systems defined in algorithm 2.1. The derived problem can be specified in natural language by the following description: The function InitialItemRecommendation is implicitly given by the offsetting module and query. Additionally, the initial candidates are passed to the module. The underlying problem of the conversational RS consists of defining the remaining fundamental functions GenerateCritiqueOptions, UpdateFilter, ApplyFilter and SelectNewItem.

The basic idea of the conversational module is to realize either of the critiquing schemes. Here a procedure should be chosen which, minimizes the path length starting at the entry point given by the offsetting module.

Implemented Solution Approach

In general, for solving the above defined problem, all critiquing schemes are possible solutions. Here the algorithms introduced as basic approaches in subsection 2.4.4 are the ones best researched with respect to efficient recommendation dialogs. These are static unit critiquing, dynamic critiquing in its Apriori-based and MAUT-based implementation, as well as incremental critiquing. This space of solution approaches is extended by the novel, hybrid schemes defined in 2.4.5.

The conversational module of the BFRS is implemented as an Apriori-based, dynamic critiquing RS. This decision in favor of dynamic critiquing is motivated by several observations: First of all, static unit critiquing is provably less efficient with respect to session length and offers a less intuitive interactional interface to the user. This renders it less appropriate for the given requirements of the application domain. Secondly, MAUT-based RS may offer more efficient navigation in several domains, yet do not give the same overview on the available items. This is caused by the fact that the derived

critique options only resemble one item. On the other hand, Apriori-based compound critique generation as a form of explication on the item space (see 2.4.5) offers a method to structure the set of candidates efficiently from the user's point of view. This is especially relevant for non-expert users and offers the necessary insight into the alternatives. Thirdly, the decision against incremental critiquing is motivated by the prototypical nature of the BFRS; that means, in this work we focus on a solid baseline proving the general concept of the hybrid architecture and leave the potential extension to incremental critiquing and research on its applicability for the given domain to future work.
Finally, with respect to the novel, hybrid critiquing approaches, the implemented conversational RS works based on a similar principle compared to user adaptive ConvRS (more precisely the Adaptive Place Advisor [26]), while including the learned user profile differently. The other hybrid approaches are again extensions to dynamic critiquing and were not considered for the same reasons as incremental critiquing, especially as their improved performance has not been as well researched as for the classical approaches.

The adjustable parameters of the implemented conversational module are the criterion according to which a subset of the generated compound critiques is selected for being presented as critique options. Additionally, the number of presented compound critiques and overall critiques can be adjusted. This property would be influenced by the future chat interface of the surrounding software application, but, as it is not defined for the current system, has to be evaluated independently. Finally, the threshold for minimal support of the Apriori algorithm underlying the critique option generation has to be set.

4 Evaluation of the Conversational Recommender System for Business Flight Traveling

In the previous chapter we have introduced the overall approach and architecture of the Business Flight Recommender System based on the restrictions and requirements of the given problem domain. We further specified the modules and the chosen, most promising algorithms implementing them in the context of the architecture. In this chapter we will present the results of the evaluation of the BFRS showing the differences between multiple alternative implementations.

For this purpose we will first give an overview on the dataset underlying the evaluation, the necessary processing steps and its final quality. Having the implied restrictions of the dataset in mind we will describe the methodology of the evaluation with respect to the goals, quality measures and overall process. Finally, we will give an overview on the results and interpret them in the context of the given problem domain.

4.1 Methodology and Dataset of the Evaluation

In this section we will give a description on the underlying dataset and the methodology of the evaluation. Together this renders the basis for the adequate evaluation executed for this work.

In a first step, we will give a brief overview on the given, underlying raw data and the desired dataset and afterwards describe the preprocessing pipeline to perform this transformation. Afterwards we will focus on the methodology of evaluating the various components and implementation alternatives based on the pre-processed dataset.

4.1.1 Preprocessing Steps and the Evaluation Dataset

As the domain of business flight traveling or flight journeys in general are rare application domains in the field of research on recommender systems, there is no publicly available data set adequate for the purpose of evaluating a knowledge-based recommender system. Therefore one of the major tasks of this work was to generate a representative dataset useful for evaluating the RS from limited knowledge sources. We orient the process of preprocessing towards the general knowledge discovery process model and terminology described in [25] to receive a description consistent with the literature.

In this subsection we will first describe the raw dataset as an input of a preprocessing pipeline to receive a valuable evaluation set of data points. In a second step we will describe the desired qualities of an evaluation data set as the result of the preprocessing steps.

Description of the Raw Dataset

The raw dataset is given by a database of past business flight journey bookings from the years 2013 to 2017. Each data point is specified in a semi-structured manner, concretely via XML-format, and consists of multiple optional and mandatory fields. Relevant to this work are the mandatory fields uniquely identifying the user account, the point in time and the flight journey of the booking. However, the storage schema changed overtime resulting in the fact that there exist many instances deviating from the currently defined standard. Additionally, the data set is not directly accessible due to privacy restrictions and has to be anonymized based on the standard schema beforehand.

Goal of the Data Preprocessing

The goal of data preprocessing is to generate a valuable dataset both reflecting authentic user behavior representative for the domain and given in an adequate format for the evaluation of the components and overall system of the Business Flight Recommender System. The goal format for each data point should contain at least the following fields for their named purposes:

Traveler Identifier A unique identifier for each entry has to be given. This allows to separate the set of all entries into groups, where one group can be associated with one traveler or rather one user.

Booking Date and Time The date and time when the booking was submitted are necessary. Hereby the database entries can be sorted in the authentic, timely order.

Query The initial query submitted by the user, which expresses his short-term needs at the point of booking, is required. This allows to involve this information for the evaluation including the query-based RS.

Candidate Set The set of candidates returned by the flight retrieval engine in response to the previously mentioned query is necessary. This allows the evaluation of the BFRS based on candidate sets representative for the surrounding software application.

Booked Item The item which was finally booked is, of course, required. This flight journey is selected from the set of candidates of the entry. Hereby the deduction of a timely ordered history of positive feedbacks is possible.

Considered Non-booked Items Lastly, the ideal dataset stores the items of the candidate set, which were considered by the user during the selection process. This may be represented by a simple sequence of item considerations. This allows the evaluation based on an authentic estimation of implicit, negative item feedbacks.

It is apparent that the given raw data set does not offer all of the above fields for each data point and needs to be preprocessed for evaluation purposes. As described in the paragraph on the raw entries, only the unique user identifier, the date and time of booking, as well as the booked item are given. For this purpose the flight retrieval engine as an additional knowledge source has to be considered, allowing to submit queries for future journeys and recording input and output. From the restriction of querying only for future dates follows the necessity for a sophisticated *data integration* step, that is the combination of multiple knowledge sources [25], during preprocessing, which we will describe now.

Preprocessing Pipeline and Final Evaluation Dataset

The preprocessing pipeline employed for receiving a valuable dataset according to the above goal description consists of five steps defined by the following sequence of preprocessing measures.

1. *Anonymization:* Due to the presence of privacy sensible fields in the description of the bookings in the raw dataset any personal information are removed in this step without losing the information which bookings share the same traveler by introducing pseudonyms. The resulting dataset is also reduced to contain only those data points with a valid description according to the current norm due to the design of the anonymization process. This step is necessary as the data is not available to this work in a non-anonymized form.

2. *Dataset Filtering:* In the second step the anonymized dataset is reduced to a subset of entries by the following criteria: Every user and his entries with more than 200 bookings per year (ca. 4 bookings per week) are discarded, as a maximum of 4 journeys per week is a realistic upper boundary for the business flight traveling domain according to domain experts. Users having a number of bookings above that threshold probably are the result of booking for multiple different people over the same account. This reflects no authentic user behavior according to the assumptions in 2.2.1. Every user and his entries with less than 10 bookings are removed, as well, because the adaptive algorithm needs a minimal number of feedbacks to be evaluated properly. The number 10 was chosen, as during the evaluation each user's entries are split into training, development and test data. Here the test split consists of 20% of the data (see 4.1.3) resulting in at least 2 entries per user, which seems a reasonable lower boundary for an evaluation with the given data. Finally, all bookings with invalid feature values, caused by the inconsistent storage schema over time, or those deviating from a common standard are excluded. This step is a form of what García et al. [25] define as *data cleaning* and *normalization*.

3. *Query Deduction:* After the previous step only users with a valid booking behavior and entry formats are given. From every booking a structured query is deduced as the query matching all abstract fields of the booked journey. However, as the booking lies in the past, the queried points in time are determined by adding the difference between the date of submitting the booking and the date of the journey. The resulting date is further offset to the closest weekday in the future matching the weekday of the booked journey. This lies close to an authentic query.

4. *Candidate Retrieval:* The query deduced beforehand is handed to the flight retrieval engine and the resulting candidate set is stored in the dataset together with the original booking, query and meta-data. Together with the previous step this process is a form of *data integration* [25].

5. *Booked Item Mapping:* In a last step, an item most similar, i.e. according to a similarity measure defined together with domain experts, to the booked item (again mapped to the future point in time as in step 3) is chosen from the candidate set. All other items of the candidate set are considered instances of negative feedback.

This preprocessing steps together with the resulting data fields before and after each step are depicted in figure 4.1.
The final dataset hereby consists of data points with all the fields necessary for evaluation. Starting from the anonymized database with nearly 17 thousand entries (i.e. bookings) the final dataset contains ca. 1.6 thousand data points. This great reduction appeared in the filtering step to ensure the dataset quality with respect to authentic user behavior and validity of feature descriptions of bookings. Additionally, the requirement for a minimum of samples per user for testing the various training constellations of the adaptive long-term preference RS lead to a decrease in the amount of data.

In the final dataset there are 63 users. With every user on average 23 bookings, but at least 10 and at most 38 can be associated. Considering all candidates in the dataset, that means *with duplicates*, there are more than 280 thousand journeys available. Erasing duplicate journeys according to their most abstract features (i.e. departure and arrival locations and points in time) a total number of about 220 thousand journeys remains. These journeys are composed upon ca. 92 thousand distinct flights (by their abstract features), which can be reduced to 5.2 thousand flight connections according to their distinct flight numbers

Quality of the Evaluation Dataset

The generation of the dataset involved several deduction steps using heuristics supported by the knowledge of domain experts. Nevertheless, these transformations may introduce noise and biases, which have to be kept in mind during evaluation. Now we will give a short overview on the quality of the resulting dataset with respect to these potential shortcomings.

First of all, the dataset guarantees that each booking is described by a common schema implying the same set of mandatory and optional feature information for each item. This normalization makes the selection of queries and best candidates more appropriate and reliable. Further on, users with clearly invalid booking histories are discarded reducing the noise in the dataset.
On the other hand, in the third and fifth step the query and item, respectively, are chosen based on a similarity measure. An essential requirement for the validity of the selection is therefore that the similarity measure reflects the user's perceived similarity. This can only be approximated, which inherently results in a less authentic data set.
Additionally, one can argue that, in general, the feature-based selection of the booked item could favor knowledge-based approaches, as no unobservable information, like e.g. security statistics on airlines, are taken into account. Yet by the assumptions that the central implicit features of journeys did not change drastically over the years, in which the booking data was recorded, the similarity-based selection should directly transfer this implicit preferences to the current date. As the raw dataset is quite recent containing entries of the past 4 years, we argue that this assumption is adequate.
With respect to the candidate set deduced from the query one could argue that it could deviate significantly from the past candidate set, because either the deduced query in the third step is not completely authentic or the structure of the offered flights could have changed compared to the past. Again due to the recency of the data, we argue that the impact of the change in the offered flights is not too drastic. The deduced query we argue that it is sufficient to imply the mandatory fields, which are the essential information revealing the context of the user, with high accuracy. This is done quite reliably for the queries by an adequate heuristic in the step three of the pipeline.
Finally, the selection of data points meeting a standard schema and those users with at least ten bookings may introduce a bias to the dataset. Here one obvious flaw is that casual users, which lie in the scope of the BFRS (especially the conversational component), as well, are discarded.

We draw the conclusion that the generated dataset is not flawless, but offers a solid evaluation basis. We argue that the results of the evaluation on this data should be interpreted with caution with respect to the absolute metric values. However, the results render a solid basis for assessing the overall concept of the BFRS and for comparing various algorithmic approaches for the components. Additionally, the resulting numbers of the evaluation, described in the following section, indicate that the preprocessing pipeline, to a certain degree, sustained the authentic user behavior implied in the raw data, which goes beyond decision making by pure chance and enables actual learning; one might refer to the random baseline results in 4.2.1 and the learning component's results 4.2.2.

4.1.2 Goals of the Evaluation

As Herlocker et al. [28] point out there are various goals and settings for evaluating recommender systems with respect to different quality criteria. In this subsection we will give a brief overview of the relevant evaluation goals and implementing methodologies for the Business Flight Recommender System.

We will discuss the potential evaluation goals compliant with the various quality properties of the BFRS described in 3.1.3. Therefore we will first focus on evaluating the recommendation performance and afterwards the degree of user comfort.

Evaluating the Recommendation Performance of the Business Flight Recommender System

We understand the recommendation performance to describe the ability of a given recommender system to make correct recommendations. Evaluating the recommendation performance offers insight in the theoretical accuracy of the underlying recommendation algorithm abstracting from the other components of the RS. It is noteworthy that this property of a recommender system is intrinsically user-centered disregarding interests of other stakeholders.

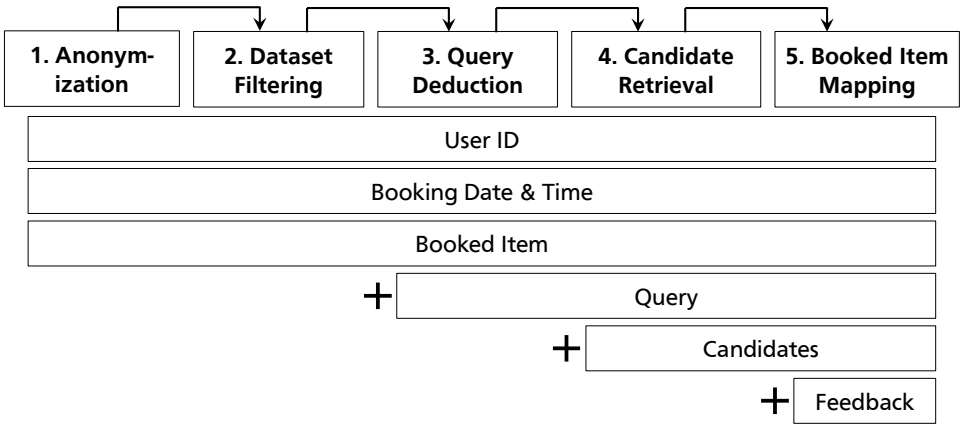


Figure 4.1.: The five steps of the preprocessing pipeline where the fields of the database entries after each step are listed in the vertical column below the step. The pluses indicate the adding of a derived field.

In the subjective domain of preferences the notion of correctness is a multi-faceted term and can be measured by various accuracy metrics from the field of information retrieval and machine learning [28]. The typical evaluation scenario for measuring the recommendation performance is a so-called *off-line analysis on historical datasets* [31]. Here item, user and preference data acquired in the past are used as an input for the recommendation algorithm. The resulting recommendations, predictions or rankings are then compared to a withhold proportion of the user preference data to determine the degree of compliance.

As the goal of evaluating the recommendation performance is fundamental for determining the quality of a RS this is the central goal to the evaluation of the Business Flight Recommender System. Additionally, for the prototype only off-line data is available, which renders this form of evaluation possible.

Evaluating the User Satisfaction of the Business Flight Recommender System

The user satisfaction of a recommender system describes the degree of perceived usefulness offered to the user or the required effort imposed on him during the recommendation process [31]. This property is evaluated to determine the potential of the overall recommender system to be used and, in e-commerce scenarios, generate business value.

For evaluating the user satisfaction usually so-called *on-line experiments* and *user studies* using questionnaires are used [31]. In the former scenario the RS is employed inside a real software application or service offering insight in live usage data. Alternatively, off-line evaluations focusing on metrics related to the problems in RS (refer to 2.2.7) can be considered as settings for analyzing the user satisfaction: Here, for example, the item space coverage, degree of novelty of recommendations or the user confidence in the RS can be estimated [28].

For the BFRS, as no on-line data is available due to its prototypical nature, no formal evaluation of the user satisfaction is carried out. The off-line evaluation with respect to the challenges in recommender systems focuses on the new user problem, as this is highly relevant considering the heterogeneous user base and the flexibility of profile adaption, due to the aim-oriented nature of user desires.

In the following subsections the methodology of the evaluation of the Business Flight Recommender System will be described with respect to the various goals. The evaluation process relies on the off-line dataset described in the previous subsections. To determine the impact and value of each of the components, hereby validating the design described in the previous section, the evaluation is split into four steps: First the existing system as a baseline, then the offsetting module, thirdly the conversational module and finally the complete system is evaluated.

4.1.3 Evaluation of the Offsetting Module

In the previous subsections we have described the basis for the evaluation of the Business Flight Recommender System. Now we will give a brief overview on the methodology for evaluating the recommender systems inside the offsetting module and finally the overall parallel hybrid.

We will first define the relevant recommendation performance metrics, secondly describe the separation of the evaluation dataset and finally define the evaluation setup for each component.

Recommendation Performance Metrics

As the offset recommender system is used to determine the entry point for the conversational recommender system, performance metrics on the recommendations are the most relevant. This opposes the approach using metrics for estimating how accurate the rankings or predictions by the RS are [31]. Typical metrics for this purpose are *precision*, *recall*, *F1-measure* and *accuracy* of recommended items [28]. Refer to table 4.1 for a formal definition for the domain of recommendations. For every metric a *top-k* variant may be considered, which means that from the ranked item list returned by the RS the top $k \in \mathbb{N}$ items are considered to determine the metrics.

Analyzing the recommendation performance using the above metrics allows to optimize each of the components' parameters both in isolation and combination. Hereby the contribution of the offsetting module and its components can be determined to prove the design decision, described in the previous chapter, correct. The top-1-precision of a RS specifies the probability of reducing the critiquing path length to zero, if the respective module was used as an offsetting system and is hereby the most important one. This is the case, because the entry point for the dialog, determined as the top-1 item, is already the best match (i.e. no critiques are required) with the probability given by the top-1-precision. Nevertheless, we will often focus on top-3- and top-k-precision for other values of k, as well. Additionally, top-1- and top-3-F1-measure consider the recall, as well, indicating the balance between these individual scores.

Metric	Definition	Intuition
Top-k-Precision	$\frac{TP}{TP+FP}$	The probability of one recommendation of the list of k recommended items to be valuable to the user.
Top-k-Recall	$\frac{TP}{TP+FN}$	The probability of a valuable item to be recommended in the top k of the list of recommended items.
Top-k-F1-Measure	$\frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$	Harmonic mean of top-k-precision and top-k-recall.
Top-k-Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$	Probability of the RS to classify an item correctly as valuable or invaluable by the top k ranking.

Table 4.1.: Typical metrics for the evaluation of RS. TP/TF denotes true positives/negatives, FP/N false positives/negatives. In RS a positive is each element of the $k \in \mathbb{N}$ top items of the ranking while the true positives and false negatives are defined by the set of valuable items for the user.

Separation of the Evaluation Data

For the evaluation of each of the components and the overall module the dataset is split by a proportion of 0.7 to 0.1. to 0.2 into training, development and test data, respectively. This ensures a minimum of 2 test samples per user, as every user has at least 10 entries in his booking history. In total, for all 63 users, this makes at least 126 test samples. This renders a reasonable test set of above 100 samples considering the given underlying data. For components lacking the necessity of learning, the training share is also considered for development. It is noteworthy, that the evaluation dataset is split user-wise, meaning that for every user's history of booking entries the above split is executed. See figure 4.2 for an overview on the split underlying the evaluation of all components of the RS.
Additionally, as the underlying data has a timely order, which may have significant influence on the recommendation performance, the history of each user is kept in the given order. Whereas each candidate set per entry is randomly perturbed. Finally, the evaluation on the test dataset was, of course, only executed after the final system configuration was determined in the development phase to identify the value and impact of each component.

Evaluation Setup of the Query-based Recommender System

For the query-based RS the metrics are determined by the following steps on each entry: Firstly, the query and candidates are given as an input and the recommendation list is generated by the RS. Secondly, the result is compared to the actual feedback stored within the entry and the counts for the metrics are updated.

During development the underlying utility function can be tuned (refer to 3.3.4). Here for all variants a function leaned onto weighted hamming similarity measures was implemented, identical in the local similarity measures, but with different weights of the sub-queries (α_1, α_2) and local measures (ω_i). In every development step only either sub-query weights or the local measures were altered and afterwards compared. In total twelve configurations were examined. As a starting point for each type of weight uniform factors were chosen and later adjusted.
Additionally, all experiments were executed with both the full query stored per entry and a reduced variant containing mandatory fields, only. See table A.3 for a complete listing with designators for each combination.

Evaluation Setup of the Long-term-preference-based Recommender System

The RS based on long-term preferences is evaluated by the following steps: For every user the subsequence of training data ordered by time is selected and used as a training input for the adaptive algorithm. Hereby for every user a new profile is learned, which is then exploited for the generation of item rankings for all entries of the development or later the testing subsequences of the respective user history. The metrics are updated for every such entry comparing the resulting list with the actual feedback.
To account for the evaluation goal of estimating the performance of the RS with respect to the new user problem, for every configuration the performance is measured when trained on all or the last (i.e. newest) 15, 10 or 5 training entries.

In the development phase two central parameters have to be determined: The feature representation of journeys and the mechanism for handling the imbalanced feedback classes for each entry. First a reasonable feature representation of journeys was identified in multiple experiments using random undersampling of the majority class, where from each candidate set only one negative item was randomly chosen and added to the training data together with the booked item. Finally, four sampling methods with different proportions of minority and majority class items and different generation mechanisms were considered.
Additionally, for reference purposes a simple k-Nearest-Neighbor classifier using similarity weighting was implemented and evaluated in three constellations for $k = 3, 5, 10$. See table A.4 for an overview on the nine development setups.

Evaluation Setup of the Complete Offsetting Module

The complete offsetting module is evaluated by blending the previously described setups into one: The data is considered user-wise to adapt the long-term preference RS based on the training share of each user individually. After the training phase both RS receive the candidates as an input, while the query-based RS also takes the query, and generate their results. The final result, after the hybrid combination of the results by the respective function, is again compared to the actual preference data to update the metrics for each entry of the development and later test set.

To determine the best parallel hybrid design the best performing configurations of the subordinate RSs are considered as fixed and only the hyper-parameters of the combination function are tuned. This involves the setting of the threshold and normalization factors. Hereby the computational complexity of the evaluation is greatly reduced compared to the development tuning all parameters together, although this might lead to a non-optimal solution. Nevertheless, for the prototype this procedure is definitely sufficient. During development twelve hyper-parameter and combination scheme constellations were tested, which can be found in table A.5. Additionally, like before the training data is reduced to smaller sub-sequences to determine the effects of few samples on the recommendation performance.

4.1.4 Evaluation of the Conversational Module

The implementing critiquing-based recommender system of the conversational module cannot be evaluated with classical recommendation performance metrics like they were used for the offsetting module. This is the consequence of the iterative recommendation approach. In this subsection we will introduce to usual method employed in the field of research on CBRS for evaluating such systems.

Firstly we will first define adequate metrics for determining the quality of recommendation sessions in CBRS, specify the dataset separation and afterwards discuss the evaluation setup for the implemented conversational module in accordance to the literature.

Evaluation Metrics for Critiquing-based Recommender Systems

Critiquing-based recommender systems as a special form of conversational recommender systems do not offer a single-shot recommendation (ref 2.2.9), but a sequence of propositions usually terminating in an accepted item. As a consequence metrics determining the accuracy of a single recommendation are not very useful to measure performance or user satisfaction. Therefore, the usual metric for CBRS is the *efficiency of critiquing sessions* calculated by the average critiquing path length [34]. Additionally, the so-called *critique prediction accuracy* describing the proportion of certain

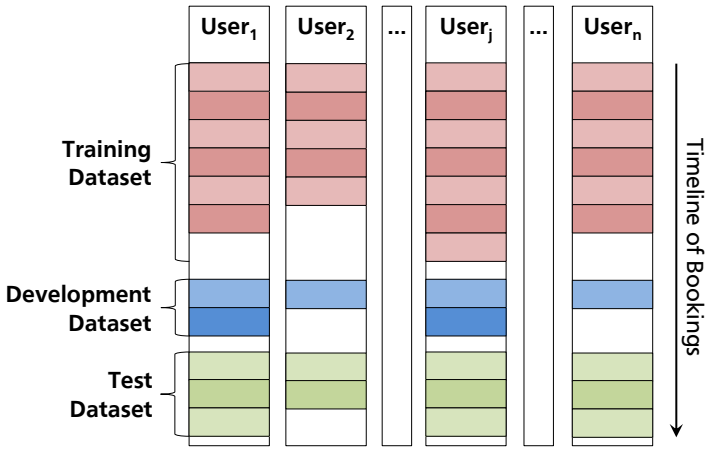


Figure 4.2.: The data split done before the evaluation. Each user's booking history is split into 0.7 training, 0.1 development and 0.2 test data. When talking about the data sections, if not explicitly stated otherwise, the conjunction of all entries of the respective sections of all users is referred to ignoring the underlying structure.

critique options (e.g. compounds), that where selected, compared to all generated critique options. This is a reasonable measure to predict the value of generated critique types. For the case of dynamic compound critiquing this is the number of compound critiques selected divided by the number of all critique options presented. Refer to table 4.2 for an overview on the employed metrics and their formal definition.

Separation of the Evaluation Data

The data is separated by the same procedure described in subsection 4.1.3 on the offsetting module. While for the long-term-preference-based RS it was necessary to distinguish between users, for the conversational component the resulting training and development sections for all users are put together and randomized after the user-wise split. As timely considerations are not relevant this prevents potential biases for entry subset selection while adjusting the parameters, while maintaining the split structure.

Evaluation Setup of the Conversational Module

The evaluation setup is defined in accordance to the typical off-line evaluation scenario described for critiquing-based RS in the milestone papers of this field of research [45], [46] or [35]. This scenario involves the following steps applied for every database entry, where the resulting metrics are averaged over the considered ones:
By a leave-on-out strategy, iteratively one of the items of the candidates is stated as the *most preferred item* of an *artificial user*. This item is excluded from the candidates and the journey most similar to it is determined as the *aim item* of the session. Here the most preferred item is understood as the preference model of the artificial user, which however cannot be met completely, as it is not in the set anymore. The aim item therefore is the journey best fitting his needs and will be chosen. Now the remaining candidates (in a randomized order) are passed to the critiquing-based RS, which starts a new session. The artificial user responds by selecting a critique option from the offered ones, that matches the excluded, most preferred item or accepts the proposition, if it is the aim item. This process is repeated for each candidate in the set and the resulting metrics are averaged over all sessions. Additionally, a threshold for aborting a session at a maximum number of cycles was added for practical reasons. Here the value of 300 cycles per session was set to account for the about 200 candidates per set leaving some room for redundant navigation due to the artificial recommendation scenario.

This evaluation approach is definitely not flawless, as, for example, it is assumed that all users desire exactly one item from the catalog. Unsatisfiable users or users open for multiple recommendations are not considered. Additionally, the human selection behavior of critique options or a change of preferences during the session is completely abstracted from. Nevertheless, this methodology offers a value comparable with literature results and is an acceptable procedure for off-line evaluations.
During the development phase multiple combinations of number of dynamic unit critiques and compound critiques were considered, motivated by the restrictions of the chat interface by the future application. For the best performing combinations the minimal support threshold of the Apriori algorithm was varied. Additionally, a ranking mechanism of critique options was implemented in three variants: Ranked by minimal and maximal support and random ordering. Finally, as a reference, dynamic and static unit critiquing were implemented and tested as well, to approximate the influence of compound critiques on the recommendation efficiency. Refer to table A.6 for a listing of the considered configurations.

Metric	Definition	Intuition
Average Path Length	$\frac{\sum_{i=1}^{n} pathlength(i)}{n}$	The mean expected number of cycles during a critiquing session.
Proportion Aborted Sessions	$\frac{\sum_{i=1}^{n} success(i)}{n}$	The probability of a session to terminate in a user abort.
Proportion Unit Critiques Selected	$\frac{\#UnitCritiqueOptions}{\#AllCritiqueOptions}$	The probability of receiving a unit critique as a critique option.

Table 4.2.: Typical CBRS metrics. It is assumed that $n \in \mathbb{N}$ sessions were observed, that *pathlength* returns the path length of a session and *success* returns 1, if and only if the input session was not aborted. # denotes 'number of'.

4.1.5 Evaluation of the Complete Business Flight Recommender System

The complete Business Blight Recommender System has to be evaluated by employing a mixed setup adequate for both the long-term-preference-based and the conversational RS. Therefore, in this subsection, we will focus on the evaluation of the whole system.

As in the previous subsection we will first describe the evaluation metrics, data separation and evaluation setup for the complete BFRS.

Evaluation Metrics for the Business Flight Recommender System

As the BFRS is a form of a hybrid, conversational recommender system the same metrics as for the critiquing-based RS apply in general. The desired effect of the offsetting module is to shorten the critiquing paths by selecting the entry point to be the most preferred item or lie as close to it as possible. Therefore the average session length specified in table 4.2 is the most relevant measure of recommendation performance of the whole system. The additional metrics for CBRS reflecting, to a certain degree, user comfort may be considered, as well.

Separation of the Evaluation Data

The evaluation data is separated as described in 4.1.3 keeping the timely order per user. Because the complete system does not require any development anymore, as this was done by individual adjustment of each component, the development split of the data is not be considered.

Evaluation Setup of the Business Flight Recommender System

The evaluation setup combines the scenarios for the offsetting and the conversational component. The process of evaluation consists of the following steps executed for each user's booking history: First of all, the offsetting module is trained on the training data as described in 4.1.3. Before applying the offset module to each entry of the test share the mapped booked item is removed from the candidate set, as this will be necessary for the CBRS evaluation. The hereby determined items are used to find an entry point for the conversational RS. For the conversational component a similar procedure to the one described in 4.1.4 is applied with the following adaption: The best item of the user is set as the mapped booked item, which is already excluded from the dataset. The aim item is than again determined as the most similar journey left in the reduced candidate set. Finally, the critiquing session is executed with this artificial user.

This setup, like the one for the critiquing-based RS, possesses certain flaws with respect to the estimated user behavior. In this setting the choice of the desired item might be more appropriate due to the fact that not all items are treated as desirable during the evaluation, but only those chosen by a real user. However, one can argue, that the long-term adaptive component learns the behavior of the user very well and would correctly predict the entry point as the desired item, but might have bad ranking performance; i.e. it has a high top-1, but low top-3 or top-5 performance. Then the second recommended item of the ranking, which is now the first, is a bad recommendation. Hereby the ability of the offsetting module to determine an entry point is biased towards algorithms offering better rankings, while in reality better single-shot, single item recommendations are preferable. Nevertheless, this is the most adequate methodology feasible for the given circumstances. This aspect is important to consider during the intepretation of the evaluation results.

The evaluation on test data will be carried out in the final constellation of the recommender system and its components. However, the effects of the offsetting module and its components are relevant to this work. Therefore configurations with the long-term-preference-based and the query-based RS are turned off will be considered, as well. Additionally, the impact of the number of learning samples available is again determined to estimate the gravity of the new user problem

for the whole system. Lastly, as a reference, for every constellation described above a static unit critiquing, as well as the final dynamic compound critiquing approach are considered. Refer to table A.7 for a list of all configurations tested.

4.2 Evaluation Results

According to the setup based on the evaluation dataset described in the previous section an evaluation was executed to estimate the overall quality of the BFRS and the impact of each component to the recommendation performance. In this section we will describe relevant, selected evaluation results and interpret them before the background of the domain analysis and system design described in chapter 3. We will show the the designed system is a very valuable baseline and the assumptions on the system parameters and their mapping to recommendation modules are adequate.

For this purpose we will first define two baselines for the contextualization of the evaluation results. Afterwards we will highlight results of the development and testing phase of the evaluation of the components of the offsetting module, the overall offsetting module, the conversational RS and finally the complete BFRS.

4.2.1 Reference Baseline for the Evaluation

For a reasonable evaluation of the developed recommender system a baseline for comparison is required. In this subsection we will introduce two baselines, the *theoretical* and the *practical* baselines, referring to the trivial recommendation algorithm for the given problem domain and the performance of the flight retrieval engine.

Firstly we will introduce the modeling of the theoretical baseline and define its metric values, afterwards we will give an overview on the measured practical baseline for the flight retrieval engine.

Theoretical Baseline

We define the theoretical baseline system as the trivial recommendation algorithm which randomly selects items from the dataset to form a ranking for recommendation. For the given problem this means that from each entries candidate set for a ranking of length one (i.e. $k = 1$ for the metrics) one candidate is selected by a probability according to the uniform distribution. For a ranking of length above one the algorithm selects a sequence of items drawing from the set of candidates without duplicates by the same procedure to create a ranking.

Splitting the candidate set into relevant and irrelevant items, the hypergeometric distribution [52] appropriately models the probability of selecting at most y relevant items within the sequence of k drawings out of a set of N candidates containing M relevant items. By determining the expected value of the described distribution for each database entry the number of hits and misses can be determined as a basis for calculating the different performance metrics mentioned in 4.1. Then taking the average over all entries gives the expected values for the metrics. The results for various top-k choices can be found in table A.8 or in their graphical form in figure 4.3.

Practical Baseline

The practical baseline is defined by the underlying flight retrieval system, which returns a ranked list in response to a given user query. Hereby the practical baseline renders the performance of the status-quo system. The developed system should at least perform better than this baseline. The evaluation was carried out by determining the metrics for several top-k variants on the ranked list returned by the retrieval engine. The results can be found in table A.8 and again in the graph in 4.3.

The line graphs in 4.3 make apparent that the practical baseline performs better with respect to the most important metrics, precision and F1-measure, suggesting that the retrieved order of journeys is not completely random. The performance, nevertheless, is very poor considering that for the best constellation (at $k = 1$) the probability of receiving a useful recommendation (i.e. the precision) lies below 2%. From table A.8 one can see that the accuracy, as it is typical for information retrieval scenarios, is very high and reveals little on the ability of the system to make good recommendations. This can is the consequence of the fact that the relevance class is drastically underrepresented resulting in very high accuracy values for simply labeling every item as irrelevant, just like it is the case for the given setting.

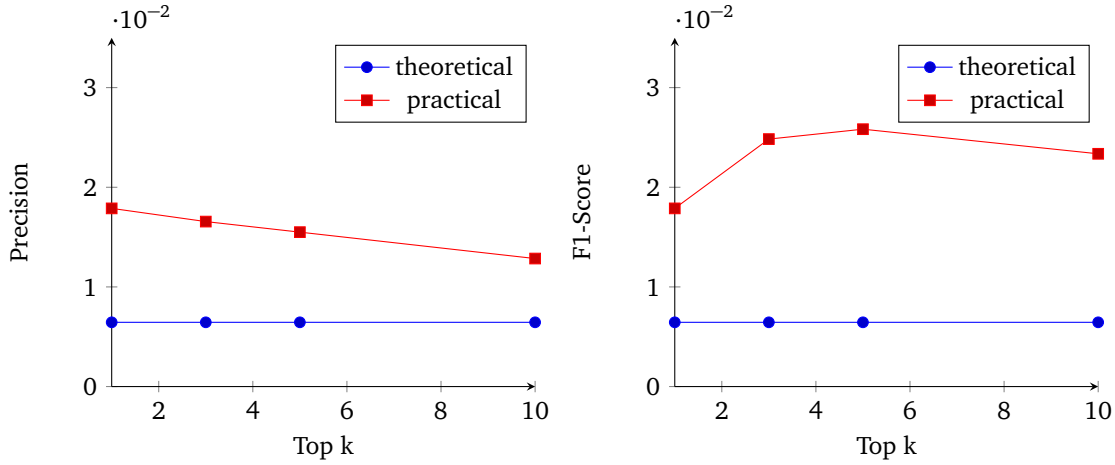


Figure 4.3.: The results of the theoretical and practical baseline evaluation focusing on the metrics precision (to the left) and F1-score (to the right) for $k \in \{1, 3, 5, 10\}$.

4.2.2 Evaluation Results for the Offsetting Module

Having the practical baseline in mind as a lower boundary for the recommendation performance, the evaluation of the offsetting module and its component can be carried out. The evaluation is split in a development and a testing phase for each component. As we will show by selecting the most important evaluation results the first module of the BFRS out-performs the baseline system significantly with respect to the considered metrics.

In a first step we will give an overview on the evaluation results of the query-based RS, afterwards of the long-term-preference-based recommender system and finally of the complete offsetting module. For each component we will discuss its performance and potential impact on the overall BFRS performance.

Evaluation Results of the Query-based Recommender System

During the development two significant experiments were performed: Firstly, the weights implying the best recommendation performance with respect to precision and F1-score were tested and selected. Secondly, the influence of the reduction of the input query to the elementary properties was analyzed.
During the adjustment of best weights four classes of local weights were considered (refer to table A.3), of which the constellation emphasizing features referring to the comfort and point in time of the journey lead to the best performance. Additionally, the sub-query weight constellation emphasizing the first itinerary (or sub-query), if queried for a round trip, grant an enhanced performance. An excerpt of the data acquired by comparing the best performing hyper-parameter constellations towards the trivial uniformly chosen weights can be found in table A.9. We figure that the emphasize on comfort features like, for example, the number of stops or the cabin class in addition to the emphasized departure date in time is a more valuable criterion for distinguishing between the candidates. The journeys of one candidate set are probably quite similar with respect to their fundamental location attributes of arrival and departure, but may vary in comfort features and the point in time of the journey. This circumstance is most likely caused by the underlying algorithm of the flight retrieval engine primarily searching for journeys transporting the user to his desired location in a relatively wide window of time around the queried date.

With respect to reduced query features the evaluation on the development dataset for neither of the configurations showed a significant change in either of the metric values. This is why this will not be considered with more detail. The later evaluation on the test dataset confirmed this property, as can be seen in table A.10, where no difference by the given decimal places is observable between full and reduced queries. This is probably caused by the fact that the deduced queries from the raw bookings cannot contain certain non-mandatory fields. An example for such fields are the hour and day windows around the departure, which are not implied by the date of the journey or submission of the booking. Hereby only few non-mandatory attributes are specified in the queries leading to the fact that the reduction has only little impact on the description of the queries and hereby the query-based module.
In the graphs found in 4.4 the uniformly weighted constellation and the best choice of weights (referred to by *12F*) on the development set are compared with respect to precision and F1-score. The better performance of the *12F* weight constellation becomes apparent in this figure. From the graphs in 4.5 the best performing constellation of the query-based RS is compared to the practical baseline on the test set. This shows, that the query-based RS is itself valuable for

generating recommendations as it offers a precision and F1-score more than double as high compared to the baseline. This is a relevant improvement for new and casual users. Nevertheless, this component alone offers a relatively poor performance, if past preference information is available. We will show this in the following subsection. This meets the expectations according to the short- and long-term model of user preferences (refer to subsection 3.2).

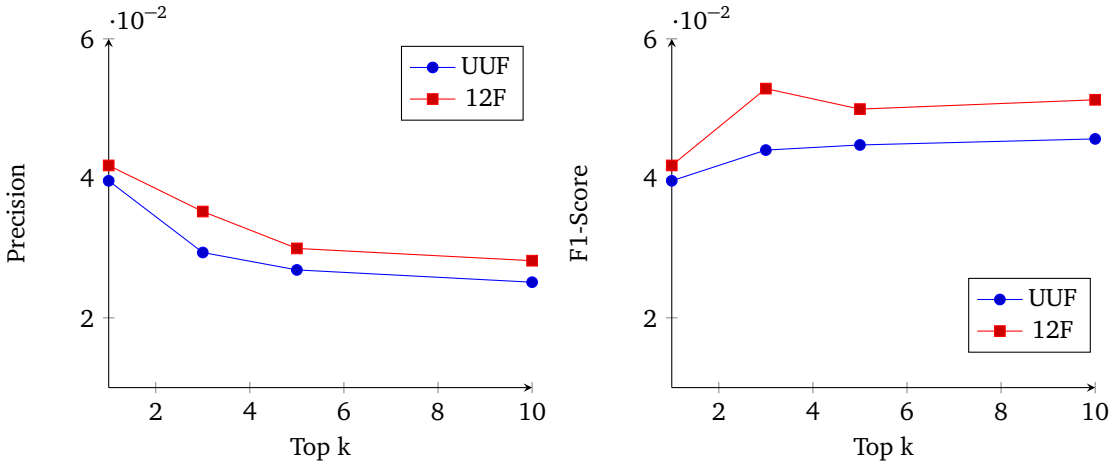


Figure 4.4.: The results of the best performing (12F) and the uniformly chosen weight (UUF) constellation for the query-based RS on the development data focusing on the metrics precision (to the left) and F1-score (to the right) for $k \in \{1, 3, 5, 10\}$.

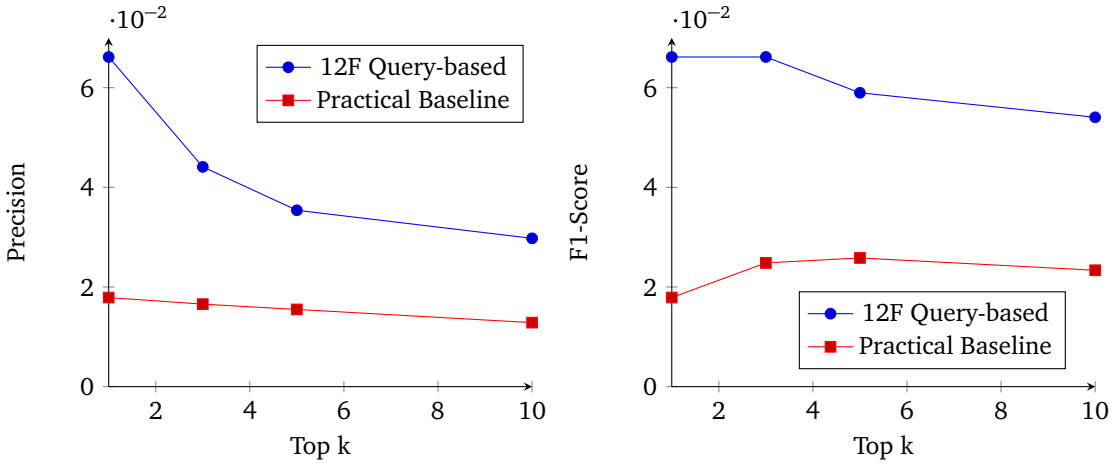


Figure 4.5.: The results of the best performing (12F) hyper-parameter constellation for the query-based RS on the test data focusing on the metrics precision (to the left) and F1-score (to the right) for $k \in \{1, 3, 5, 10\}$ compared to the practical baseline.

Evaluation Results of the Long-term-preference-based Recommender System

The evaluation of the long-term preference-based RS involved two central experiments in the development phase: Firstly, the evaluation of the impact of the sampling method considering approaches using SMOTE with under-sampling, pure under-sampling and oversampling in multiple configurations. Secondly, the comparison of performance of the Naïve Bayes classifier and the reference implementation of an k-Nearest-Neighbor classifier with simple undersampling. For all experiments the influence of a reduced history of feedbacks was considered.

An excerpt of the results of the first experiment can be found in table A.11. Here the best performing under-sampling configuration (NBU1:1) is compared to the best performing SMOTE approach (NBS3:3) for various sub-sequences of the feedbacks used for training. It becomes apparent that the Naïve Bayes classifier performs best for under-sampling of the majority class for all top-k metrics and all reduced training datasets. In this approach from every candidate set the booked item and one randomly chosen item, which was not booked, is added to the preference history with their respective labels. In a live system, unlike during the limited information of the off-line analysis, the selection of the

negative items for over- or under-sampling could be executed in a more sophisticated manner; e.g. only user reviewed items of the candidate set could be considered as negative samples.
Additionally, it is noteworthy that the performance of the classifier in its best configuration increases in the (most relevant) top-1- and top-3-precision and -F1-score for a reduction of training samples to the ten most recent ones. This is probably caused due to the fact that older feedback information is obsolete and less relevant for future predictions. This confirms the assumptions described in subsection 3.1.3.
With respect to the second experiment, comparing the best reference implementation with the best performing Naïve Bayes classifier the results of the kNN approach can be found in table A.12. From the graphs in 4.6 it becomes apparent that the Naïve Bayes classifier trained on the ten most recent samples out-performs the kNN classifier on all samples with respect to precision and F1-score. This is a strong indication for the validity of the Naïve Bayes approach for learning long-term preference profiles and exploiting them for recommendation. This is why this approach was chosen for the overall system.

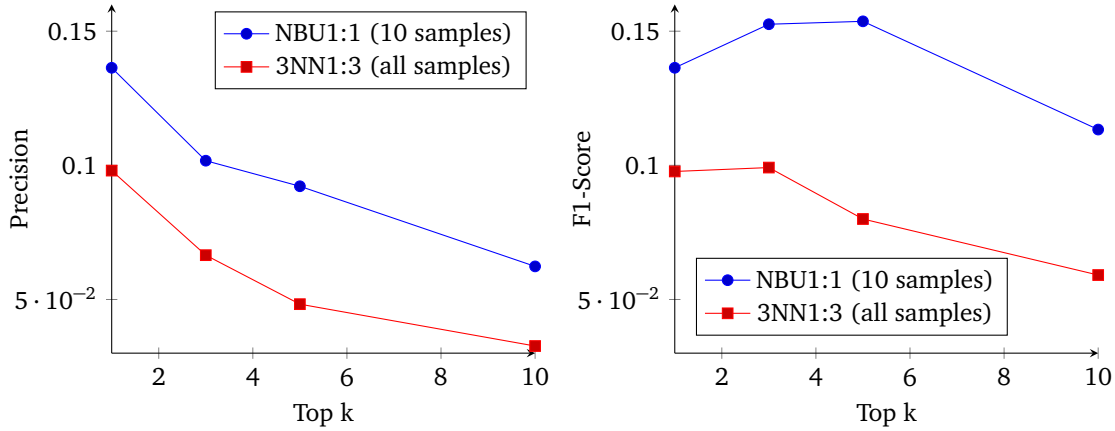


Figure 4.6.: The results of the best performing Naïve Bayes classifier (NBU1:1 with 10 feedback samples) and the best performing kNN classifier (3NN1:3 on all feedback samples) on the development data focusing on the metrics precision (to the left) and F1-score (to the right) for $k \in \{1, 3, 5, 10\}$.

Finally, during the test phase of the evaluation the results of the development phase were confirmed having only a small deviation in the values of the metrics compared to the development phase (see table A.13). There is a reasonably low generalization error of below ca. 0.02 for each metric. However, for the test set the best results are achieved for all or 15 feedback training samples. A possible explanation could be, due to the nature of the split in timely order, that the missing sequences of bookings contained in the development set, which are the most recent relative to the test instances, introduce a training bias towards older, less relevant feedback information. As a consequence, more training samples on the less relevant samples lead to a better estimation of timely independent preferences, while a reduction of training instances does not increase the quality (that is relevance) of the data and hereby lead to no enhanced performance.
The graphs in 4.7 compare the longterm-preference-based RS in its best configuration and trained on all feedback samples with the practical baseline. It becomes apparent that the model of long-term preferences is adequate and this recommendation component is per se already very valuable to travelers with a longer feedback history, that means the frequent fliers. Additionally, even for only five underlying training samples a high performance is achieved rendering the new user problem of little relevance for the given application. This property of the RS based on long-term preferences becomes apparent in the graphs 4.8 depicting the development of the recommendation performance for top-1 and top-3-precision and -F1-score over the reduced training samples.

Evaluation Results of the Complete Offsetting Module

The goal of the development phase of the complete offsetting module was to determine an adequate combination function of the results by the query-based and long-term-preference-based RSs. As can be seen from the results in table A.14 describing the performance of the best hybrid variants, the mixed hybrid composition (described in 3.3.6) is superior with respect to its precision and F1-score on the development data compared to switching hybrids. The resulting values of the metrics of the weighted hybrid lay far below the other hybrid types for all considered configurations and were hence not included.

During the final evaluation on the test data the previously described results of the long-term-preference-based RS with respect to the performance development over the number of samples were confirmed (see graphs in 4.10). There is

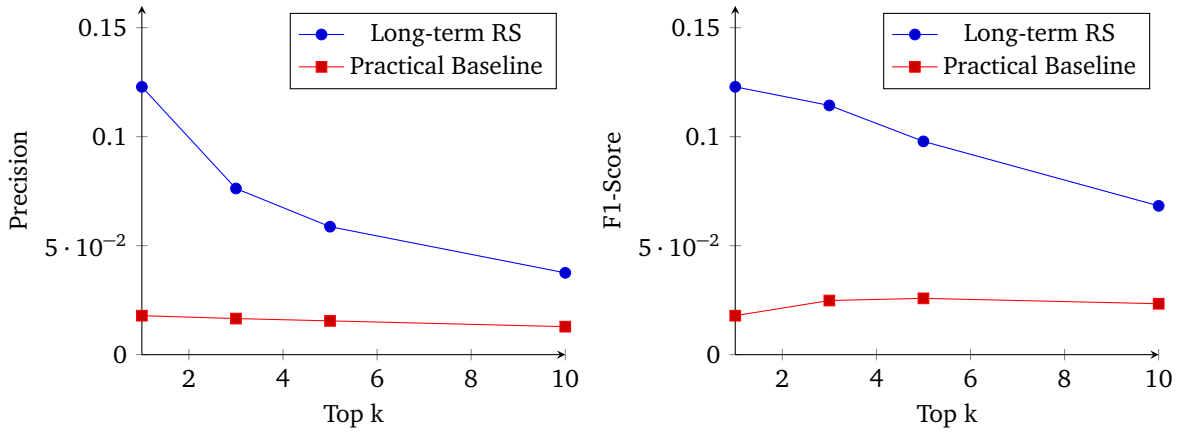


Figure 4.7.: The results of the long-term-preference-based RS on the test data compared to the practical baseline focusing on the metrics precision (to the left) and F1-score (to the right) for $k \in \{1, 3, 5, 10\}$.

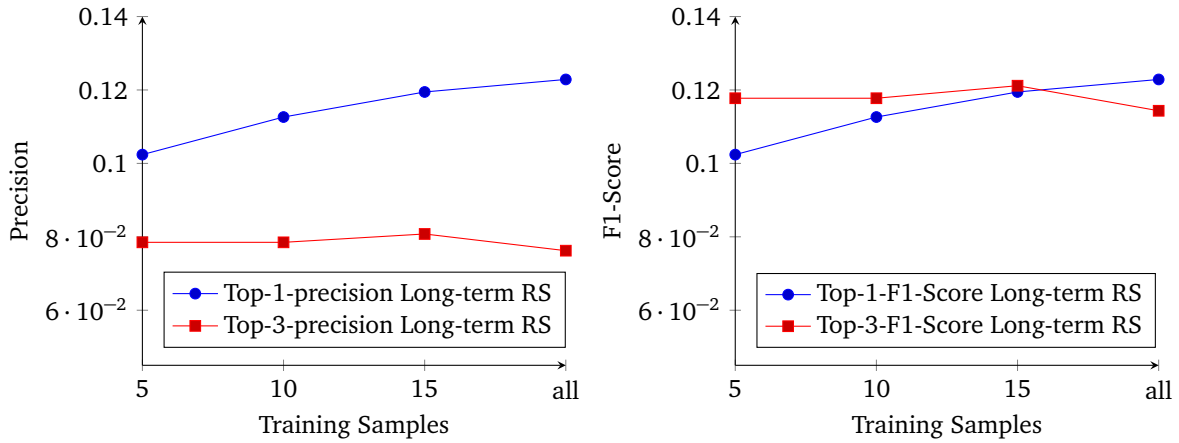


Figure 4.8.: The results of the long-term-preference-based RS on the test data for 5, 10, 15 and all most recent training samples focusing on the metrics precision (to the left) and F1-score (to the right) for $k \in \{1, 3\}$.

a small increase in performance with respect to precision and F1-score for all training scenarios and a generally low generalization error. In the graphs found in figure 4.9 the complete offsetting module trained on all samples is compared by its components' and the practical baseline's performances on the test data. It is apparent, that the offsetting module performs better than each single component, but is only little superior compared to the long-term-preference-based RS. In general the offsetting module offers itself a high value to all travelers, both casual and frequent fliers, as for $k = 1$ the probability of receiving a valuable recommendation on the first try (that is the precision) lies at about 12.5%. With a more sophisticated selection mechanism for negative samples during training, this level of performance, most likely, can be even more increased.

4.2.3 Evaluation Results for the Conversational Module

In the previous subsections we have illustrated and discussed the single-shot recommendation performance for the offsetting module, its components and the baselines. In this section we will determine the performance of various constellations of the conversational module to define a baseline for the complete BFRS efficiency of recommendation dialogs.

First we will discuss the performance of the different hyper-parameter constellations on the development data. In a second step we compare the performance of static and dynamic unit critiquing as reference to the dynamic compound critiquing implementation on the test data.

Development Results of the Conversational RS

The two goals of the development phase of the ConvRS were to find a reasonable combination of presented compound and unit critiques combined with a ranking criterion for presented critique options, which offer the best performance.

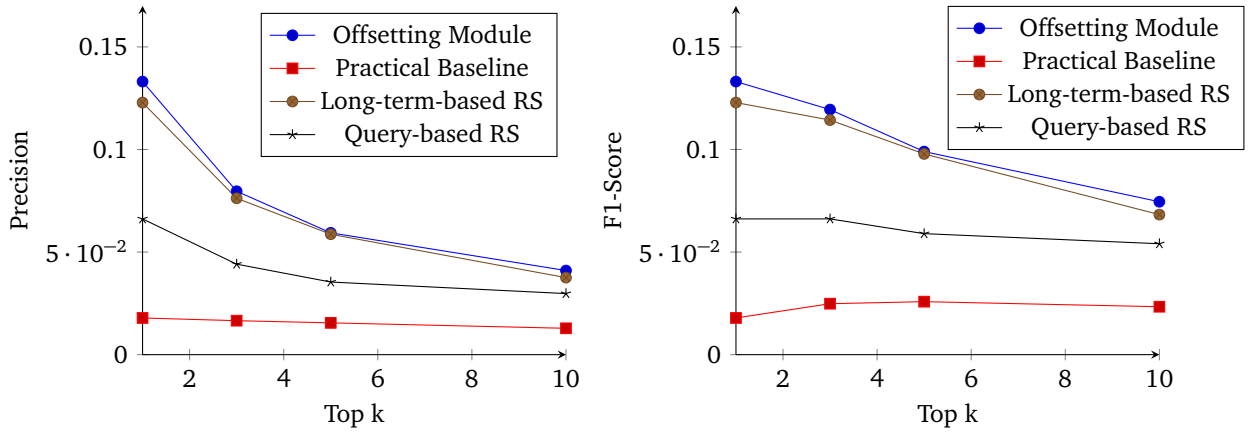


Figure 4.9.: The results of the offsetting module on the test data compared to the practical baseline and its components focusing on the metrics precision (to the left) and F1-score (to the right) for $k \in \{1, 3, 5, 10\}$.

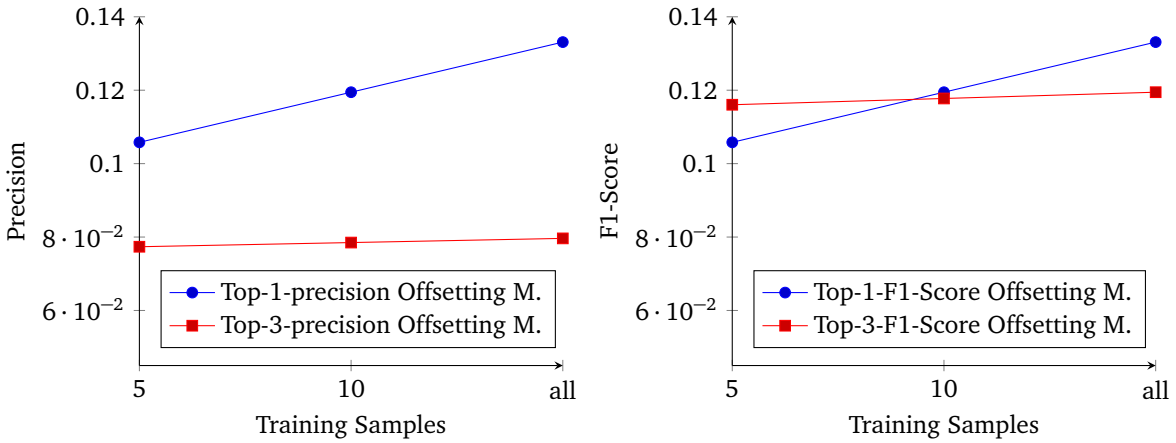


Figure 4.10.: The results of offsetting module on the test data for 5, 10 and all most recent training samples focusing on the metrics precision (to the left) and F1-score (to the right) for $k \in \{1, 3\}$.

For the various constellations of presented critiques options the results on the development set indicate that the presentation of 25 unit and 5 compound critiques is a reasonable choice (see table A.16). Here the smallest average critiquing path length is achieved at a relatively low proportion of aborted sessions. This is also the least restrictive implementation with respect to the future surrounding software application. Here basically all possible unit critique options can always be selected, as well as a small set of most relevant compound critiques.
Additionally, from table A.16, it becomes apparent that a randomized selection of critique options outperforms the minimal and maximal support selection criteria. As indicated by the increase in the average path length a minimal support between 0.1 and 0.2 results in the best performing dynamic critiquing variant. Here the proportion of selected compound critiques indicating the value of the proposed compound critique options is very high, whereas the proportion of aborted sessions is comparably high.

Evaluation Results of the Conversational RS and its Reference Implementations

On the test data the final constellation using dynamic critiquing with a threshold of 0.14 for minimal support (lying between the evaluated 0.1 and 0.2) and a randomized ranking for the critique options was evaluated. As a reference a dynamic and a static unit critiquing implementation was evaluated, as well. The results can be found in table A.17.
From the diagram in 4.11 it becomes apparent that the dynamic critiquing implementation with compound critiques outperforms the unit-critique-based implementations with respect to the average path lengths, that means the dialog efficiency. With respect to the proportion of aborted sessions this approach is the best, as well (see 4.11). This is a strong indication that dynamic critiquing is, besides its property of offering explanations on recommendations (refer to 2.4.5), more adequate for the given domain than the other critiquing approaches.

The resulting average path length of ca. 78 cycles resembles the results of Reilly et al. [46] reporting ca. 65 cycles per session on a traveling dataset. As the exact dataset and its domain is not explicitly given this comparison should be treated with care. However, it gives insight in the fact that session lengths on a complex and large dataset, like the one given for the BFRS, can lie in the high double-digit range for the given artificial off-line evaluation strategy. Yet, of course, these numbers do not directly correlate with the session length in the actual application with real users participating in the dialog. Here an on-line evaluation would be required to estimate the absolute quality of the recommender system. We are confident that for actual live experiments the results would lie significantly lower, as the preferences of real users are more dynamic over a session and are not restricted to one preferred item per candidate set.

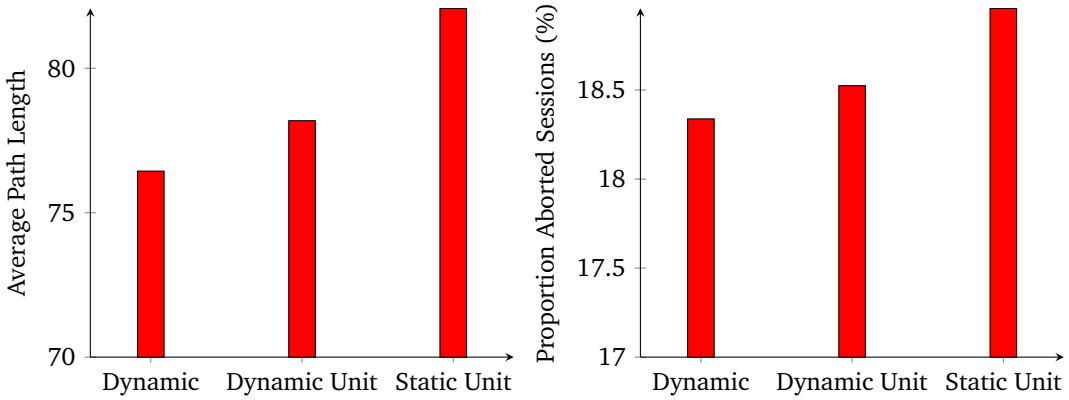


Figure 4.11.: The results of the ConvRS on the test data implemented as a dynamic critiquing approach compared to dynamic and static unit critiquing as reference implementations. For the comparison the average session length (left) and the proportion of aborted sessions (right) is considered.

4.2.4 Evaluation Results for the Complete Business Flight Recommender System

In the preceding subsections we described the development process of the hyper-parameters of the Business Flight Recommender System and analyzed the evaluation results of each component with respect to its individual and composed performance. In this subsection we will focus on the results of the evaluation of the whole BFRS contextualizing the observations on the underlying components so far. We will show that each component in the developed constellations has a positive impact on the overall performance and that, according to the off-line evaluation, the BFRS is valuable to the users.

In a first step, we will discuss the performance of three baseline and reference systems for giving the basis to identify the impact of the offsetting module and the choice of the critiquing approach. Afterwards we will introduce the results for the complete BFRS achieved on the test set comparing the performance with the previous systems and draw a final conclusion on the value of each component and the complete RS.

Evaluation Results for the Baseline System

To be able to contextualize the evaluation results of the complete BFRS on the training data it is necessary to investigate the performance of a simple baseline. As the goal is to interpret the performance of the recommender system with the offsetting module and the estimate the value of the implemented dynamic critiquing approach two reasonable baselines were evaluated (refer to the first scenario in A.7): Static unit critiquing and the dynamic critiquing approach implemented for the BFRS without any offsetting mechanisms of the entry point for the critiquing dialog. Dynamic unit critiquing was ignored due to computational complexity of the evaluation and by the consideration of the the previous results (in 4.2.3) indicating a performance between static unit and dynamic critiquing in the relevant metrics.
For these baseline systems the relevant metrics average path length and proportion of aborted sessions are depicted in graph 4.12 indicating that the dynamic critiquing approach performs best even without any offsetting module. The exact results can be found table A.18. They support the results acquired by the evaluation on the test data using the leave-one-out-strategy for every item of each candidate set described in 4.2.3.

Evaluation Results for the Components of the Business Flight Recommender System

The numbers for each system configuration leaving one component out should be interpreted in front of the background of the baseline results. For the configuration without the long-term-preference-based RS the metrics are independent of the underlying number of training samples (see table A.19). It becomes apparent that for both static and dynamic

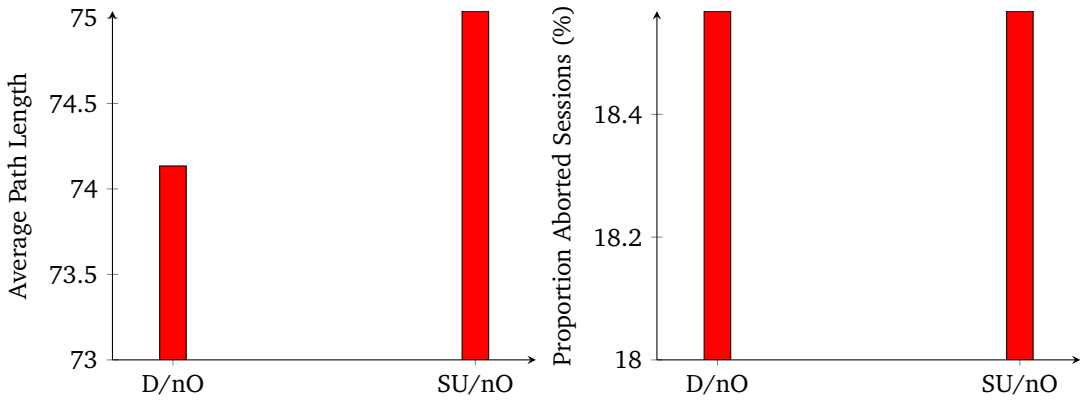


Figure 4.12.: The results of the baselines of the BFRS on the test data. For the comparison the average session length (left) and the proportion of aborted sessions (right) is considered.

critiquing a relevant performance increase can be measured. For dynamic critiquing the increase is quite large reducing the average path length by 6 cycles.
For the configuration without the query-based module the results depending on the number of training samples is shown in table A.20. Here, like for the development phase of the long-term-preference-based RS (in subsection 4.2.2) the best performance is achieved for 10 training feedback instances resulting in the shortest average path length and the lowest proportion of aborted sessions. Compared to the baseline systems for both critiquing approaches a great increase in performance of up to 6 cycles for dynamic critiquing is achieved. Comparing the pure long-term-preference-based to the pure query-based offsetting it can be said that they perform similarly well, while for 5 or all training samples the query-based RS achieves better results (see 4.13). Probably for the reasons discussed in 4.1.5, regarding the bias of the evaluation method, the performance of the long-term-preference-based RS in the case of all training samples is lower than for the stand-alone results depicted in 4.7. Here this module performs obviously better than the query-based one in isolation.

Evaluation Results of the Business Flight Recommender System

During the evaluation of the complete BFRS for various training instances the system proved to perform better with respect to path length and proportion of aborted sessions compared to the baseline (refer to table A.21). Additionally, from the graph in figure 4.13 it becomes apparent that the long-term and short-term preference modeling and recommendation are highly adequate: For few training samples the query-based RS works as a form of backup recommender system for the long-term-preference-based leading to solid results in this scenario.
Further on, the complete hybrid RS offers a greatly reduced average session length compared to each single component for at least ten samples. This is a strong indication for the adequacy of the developed approach and the concept of offsetting the entry point for critiquing dialogs. However, for very few training samples (i.e. for 5) the pure query-based RS performs better than the overall BFRS. Here a switching hybrid design would perform better, if adjusted correctly. Nevertheless, apart from statistical invariance, which is of course possible due to the relatively small test dataset, the reasons discussed in 4.1.5 could have lead to lower performance with respect to precision of the long-term-preference-based RS. The slightly higher average path length for few samples of the complete BFRS could be induced by this phenomenon.
With respect to the critiquing approach the numbers strongly support the design based on dynamic critiquing even without the considerations of user comfort offered by this approach. The static unit critiquing variant has an average path length of up to 8 cycles longer than the dynamic one (for 10 feedback samples shown in A.21). Considering the proportion of selected compound critiques, that is the inverse to the proportion of selected unit critiques, the roughly 10% given for the BFRS indicate that the compound critiques are often applicable and of actual use for the navigation through the item space.

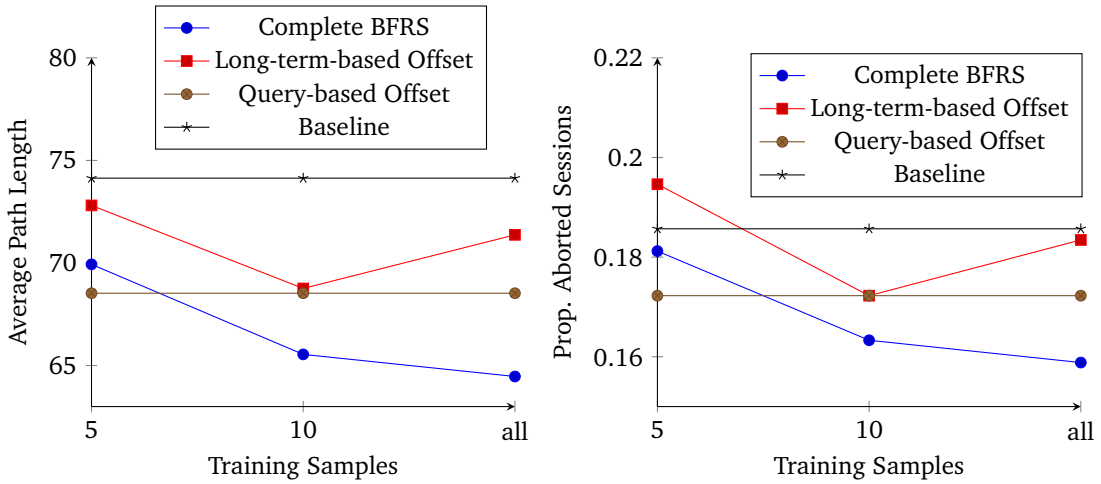


Figure 4.13.: The results of the evaluation of the complete BFRS compared with respect to the average path length (left) and proportion of aborted sessions (right) to the dynamic critiquing baseline and the components of the BFRS.

5 Conclusion

In the previous chapters we gave a comprehensive overview on the design space and a possible, most valuable implementation for a recommender system in the domain of business flight traveling. In this chapter we will summarize and discuss the results of this work, as well as the strengths and weaknesses of the implemented approach for the Business Flight Recommender System. The experiences acquired for the given data and domain will serve us as a basis for giving an outlook of this field of research and possible future extensions or alternative approaches to the given domain.

In a first step we will summarize the results of this work pointing out the most important aspects, afterwards draw a conclusion on the value of the chosen approach and implementation for the BFRS and finally give insight in possible future research and extensions to the developed system.

5.1 Summary of this Work

This work is structured in four central, interconnected steps: First we gave an overview on the relevant concepts of the research field of data mining and recommender systems including a detailed discussion of filtering techniques with a focus on so-called conversational recommender systems. Secondly, we exploited the introduced concepts as a guideline to analyze the domain of business flight traveling and deduce an adequate model of preferences as a basis for the Business Flight Recommender System. In the third step we concretized this model by giving an detailed overview on the design of the BFRS and finally, in the forth step, evaluated the implemented approach by comparing it with several reference systems and baselines.

The purpose of the first step of this work was to set a basis for defining the design space of the BFRS by establishing formal and informal models for different RS scenarios together with heuristics and guidelines for designing a RS for the employment in a live business context.
This involved a comprehensive definition of recommender systems compliant with the plethora of conceptions of RS in the literature and an analysis of their adequacy for personalization scenarios. After formalizing the recommendation problem for a clear and unambiguous separation of the concept of a complete RS and its recommendation algorithm, we defined a valuable model of components rendering the design space of recommender systems. In the end, we highlighted various aspects and the state-of-the-art extension using contextualization in RS to this basic concept of recommendation, relevant to the implemented BFRS.
Defining each relevant filtering technique formally and informally allowed us to discuss the adequacy of each approach in a scientific and application-oriented manner. Focusing on content- and knowledge-based systems we elaborated on the design space of these systems. Finally, we introduced the very suitable approach of conversational recommendation by giving a generic definition of the underlying algorithms and concepts. This overview going beyond any explanations found in the literature, so far, serves as a valuable basis for defining the architecture of the BFRS and the limits of its design space.

The second step was executed relying on the previously introduced guidelines and models in a general manner, allowing to transfer the observations on the business flight traveling market on other similar domains like train journeys or route recommendation. The subsequent analysis of the problem lead to the clear classification as a contextual recommendation problem, which is in this form a novel perspective in the field of research on contextual RS. From this observation we derived an adequate design differing between short- and long-term preferences. The defined abstract architecture is again transferable to similar domains and limits the design space of the BFRS by giving good reasons for the adequacy of the model. As proved during the later evaluation this novel modeling and design approach for similar domains is very adequate.

Afterwards, in the third step, the goal was to select implementing algorithms for each of the components. From the plethora of design options we selected a set of baseline algorithms promising a solid performance and backed the choices up by the subsequent development phase during the evaluation. The modeling of the user and item space was done in accordance with the analyzed domain parameters and the chosen algorithms.

In the final, forth step we first described the preprocessing steps on the raw dataset to receive suitable evaluation data, while pointing out its potential short-comings. For each component of the hybrid system we described an evaluation procedure suitable for development and testing, while again discussing strengths, flaws and potential biases in the off-line evaluation scenarios. Lastly, we described the evaluation results highlighting the impact of each component and

the development of the performance of the BFRS for different amounts of training samples. Hereby we proved the underlying concept and the concrete implementation of the Business Flight Recommender System to be valid and valuable for the live employment of the RS in the business context.

5.2 Conclusion on the Results of this Work

The contribution of this work is twofold: On one hand, we introduced several generic and unifying models and taxonomies for subsets of recommender systems structuring their design space. On the other hand, we developed and proved the solid performance of a recommendation approach for the concrete domain of business flight traveling, which employs an offsetting mechanism for the critiquing dialogs' entry point, which, nevertheless, can be directly transferred to similar domains like train journeys. To draw a final conclusion on these contributions we will therefore split this section in two paragraphs.

With respect to the theoretical observations on recommender systems executed for this work there are four main aspects to take into consideration: Firstly, we defined novel, unifying models and taxonomies for structuring the space of recommender systems. This included a comprehensive definition of RS considering multiple conceptions from the literature, a unifying model of components of recommender systems serving as a valuable guideline for the development of them and as a pattern for categorizing a given recommender system. Additionally, the comprehensive list of challenges in RS combining the knowledge of various sources into one list can be interpreted as a protocol for evaluating the strengths and weaknesses of filtering techniques.
Secondly, we defined a novel scheme to relate filtering techniques based on their input parameters and subsequently gave an abstract overview on them indicating their adequacy for different problem domains.
Thirdly, we analyzed and discussed the propositions for conversational RS found in the literature using again a novel taxonomy of critiquing-based RS and a list of challenges in this area of research. Finally, we analyzed the given problem domain and described its properties in a structured manner. The resulting modeling of the personalization problem in this domain as a contextual and conversational recommendation problem considering long-term and a short-term preferences can be transferred to similar domains and may be interpreted as a design blue print for RS in the field of journey and route recommendation.
All in all, the theoretical results serve as a valuable guideline for the development RS in similar item and user domains. The proved performance of the theoretical concepts for the given domain are a strong indication for their overall use to future work in this or similar fields.

The Business Flight Recommender System as a concrete implementation of the developed model employing a hybrid architecture with contextualization is a state-of-the-art conversational RS relying on the little studied approach of offsetting the entry point for critiquing-based RS. By the complete evaluation of every component of the BFRS we were able to prove not only the enhanced recommendation performance of the overall system compared to the previous one, but also that the underlying short- and long-term preference model is adequate. Apart from the user comfort and insight in the domain offered by the implemented dynamic critiquing algorithm we were able to show that the implemented components each contribute to the performance of the overall system. However, the results of the executed off-line evaluation still render a rough estimation, especially for the conversational component, requiring an on-line evaluation and user studies to have a final prove of the performance and degree of user comfort of the system. Additionally, several design alternatives, like incremental critiquing or a more sophisticated long-term preference learning mechanism, remain unexplored. Finally, the concept of treating the sequences of flights only in their aggregated form as flight journeys was well reasoned. However, it would be necessary and interesting to rule out the alternative approach relying on item sequence recommendation.

All in all, the practical results for the implemented Business Flight Recommender System prove the design and underlying modeling to be adequate and valuable for the task. The future employment of the RS inside the surrounding software application in a live business context will give deeper insight in the performance and user comfort of the system.

5.3 Outlook and Future Work

The outlook on further developments of this work's results can be divided, just like in the previous section, into the further investigation of the theoretical aspects of this work and the improvement of the Business Flight Recommender System as a concrete RS. As the BFRS was implemented as a prototype for the future employment inside a real software application a valuable on-line evaluation and the generation of an authentic off-line dataset will be available in the future making it possible to do further research on both aspects.

From a theoretical perspective the concept of employing an entry point offsetting mechanism for the conversational RS needs further research. The underlying idea is that by setting the entry point of the dialog, which is identical in its conversational flow for every user and every session, the path length is minimized or even reduced to zero by choosing an entry point close to the aim item. However, by the basic idea of critiquing efficient exploration is a core asset of such systems inducing preferences for non-experts in the first place. Meaning that not necessarily the closest entry point to some most preferred item is the best, but rather one inducing a path with few cycles which, as a whole, convinces the user to chose the final item. Here, contrary to the approach of offsetting the entry point, an adapted conversational dialog depending on previous sessions would be an alternative to investigate. Here, for example, an inter-session profile of feature importance alike the one constructed for each single session by MAUT-based dynamic critiquing approaches could be deduced from previous sessions. Further on, apart from these general research directions the further investigation of the appropriateness of the introduced module for the traveling domain in general is very promising. Experiments regarding the different levels of abstractions for describing journeys by structured features or sequences of feature vectors and their integration into a recommendation process are especially interesting.

The Business Flight Recommender System was implemented as a prototype based on limited evaluation data. In the future, when more authentic data points are available, one very important next step is to analyze whether the off-line evaluation results of this work are supported by on-line evaluations. Here both the recommendation performance, as well as the user comfort and perceived effort to interact in a critiquing dialog are very important aspects for the evaluation. Additionally, the design using the Naïve Bayes classifier for long-term-preference-based recommendation could be revised and other more sophisticated machine learning techniques tested against the defined baseline. Additionally, incremental critiquing as an improved critiquing approach could be employed and tested.
Apart from these experiments on alternative designs for the existing BFRS extensions to the RS or complete application could be implemented and tested to enhance performance and comfort of use: First of all, a module alike the one defined for the adaptive trip advisor [26], computing a best fitting query for the current session could increase the perceived comfort of use and even enhance the recommendation performance, as the cognitive effort of defining a query might introduce errors. This could also be realized by a slight alteration of a given user query using for example Rocchio's Algorithm. Additionally, the user's context could be generated implicitly or using additional features by including information like the user's location, the time of day etc. Lastly, the interaction of the recommendation algorithm with the chat layer is highly relevant especially for the conversational module strongly connected to the user interface. Therefore, experiments regarding the presentation of critique options and item propositions and their impact on the recommendation dialogs are very interesting. Here the so-called user defined critiques, which are critiques not selected from the set of options, but simply specified by the user, are especially interesting. Forms of natural language descriptions of such user defined critiques could introduce a new layer to critiquing enabling higher user comfort.

All in all, the promising results of this work should extended by further investigation of alternatives for the given domain and an extension to the underlying, theoretical concepts.

A Appendix

A.1 Algorithms

Algorithm A.1 Specification of the functions of the formal model of CBRS for the approach of static unit critiquing. All functions of the formal model of CBRS which are not mentioned here are trivial.

Require: *agr* is a measure for the degree of agreement on queries and items
1: **function** InitialItemRecommendation(q, $\mathscr{I}$)
2: **return** $\arg\max_{i \in \mathscr{I}} agr(i, q)$

Require: $\mathscr{I} = A_1 \times ... \times A_n$ is the item base
3: **function** GenerateCritiqueOptions(*curitem*, $\mathscr{I}$)
4: *options* $\leftarrow ()$ ▷ () denotes the empty sequence
5: **for** $j \in \{1, ..., n\}$ **do**
6: **if** A_j is numeric **then**
7: *options* $\leftarrow$ *options* $\circ \{(A_j, <)\}$ ▷ Compact representation of unit critiques is appended
8: *options* $\leftarrow$ *options* $\circ \{(A_j, >)\}$
9: *options* $\leftarrow$ *options* $\circ \{(A_j, \neq)\}$
10: **return** *options*

Require: *curcrit* specified by $(A_j, \circ)$ and *curitem*
11: **function** UpdateFilter(*curitem*, *curcrit*, *prevfilter*)
12: **return** $(x \mapsto curcrit(x[j])$ ▷ x[j] denotes the value of feature j of item x

13: **function** ApplyFilter(*curfilter*, $\mathscr{I}$)
14: **return** $\{x \in \mathscr{I} | curfilter(x) = true\}$

Require: *sim* is a similarity measure on $\mathscr{I}$
15: **function** SelectNewItem(*catalog*)
16: **return** $\arg\max_{i \in catalog} sim(i, curitem)$

Algorithm A.2 Specification of the functions of the formal model of CBRS for the approach of dynamic critiquing with Apriori. All functions of the formal model of CBRS which are not mentioned here are identical to the ones in static unit critiquing.

1: ▷ CritiqueFromRule is trivial and turns the feature-comparator-pairs of antecedent and consequent to unit critiques, which are completely specified by such pairs (given the current item). These unit critiques are afterwards united to one compound critique via logical conjunction.
2: ▷ Compare is another trivial function returning for two values of one of the attribute domains one fitting comparator from the set $\{=, \neq, <, >\}$.

Require: $\mathscr{I} = A_1 \times ... \times A_n$ is the item base
3: **function** GenerateCritiqueOptions(*curitem*, $\mathscr{I}$)
4: *options* ← () ▷ () denotes the empty sequence
5:
6: *patterns* ← GenerateCritiquePatterns(*curitem*, $\mathscr{I}$)
7: *rules* ← Apriori(*patterns*) ▷ Specified in 2.1.2 returning rules ranked by their support value
8: **for** $r \in$ *supportrules* **do**
9: *options* ← *options* ∘ CritiqueFromRule(r)
10:
11: **return** *options*

12: **function** GenerateCritiquePatterns(*curitem*, $\mathscr{I}$)
13: *patternset* ← {}
14: **for** *item* $\in \mathscr{I}$ **do**
15: *pattern* ← ()
16: **for** $j \in \{1, ..., n\}$ **do**
17: *pattern* ← *pattern* ∘ Compare(*item*$[j]$, *curitem*$[j]$)
18: *patternset* ← *patternset* ∪ {*pattern*}
19:
20: **return** *patternset*

Algorithm A.3 Specification of the functions of the formal model of CBRS for the approach of incremental critiquing. All functions of the formal model of CBRS which are not mentioned here are identical to the ones in Apriori-based dynamic critiquing. Leaned onto the pseudo-code in [46].

▷ GetFeedback is a trivial interface function presenting the given item and critique options to the user and returning the selected critique by the user.

Require: $\mathscr{I} = A_1 \times ... \times A_n$ is the item base; p is the current user profile as a set of unit critiques
function ProposeCritiqueOptions(*curitem*, *curopts*)
    curcrit ← GetFeedback(*curitem*, *curopts*)
    UpdateProfile(*curcrit*)
    return *curcrit*

procedure UpdateProfile(*curcrit*)
    unitcrits ← {}
    if *curcrit* is a compound critique consisting of unit critiques $(uc_j, .., uc_k)$ **then**
        for $i \in \{j, ..., k\}$ **do**
            $unitcrits \leftarrow unitcrits \cup \{uc_i\}$
    else
        $unitcrits \leftarrow \{curcrit\}$

    $p_{new} \leftarrow p$
    for $uc \in unitcrits$ **do**
        for $s \in p$, where s refers to the same feature as uc **do**
            if uc implies $s \vee uc$ contradicts s **then**
                $p_{new} \leftarrow p_{new} \setminus \{s\}$
    $p \leftarrow p_{new}$

function SelectNewItem(catalog)
    return $\arg\max_{i \in catalog}$ Quality$(i, curitem, p)$

Require: *sim* is a similarity measure on $\mathscr{I}$
function Quality($item_1$, $item_2$, *profile*)
    $similarity \leftarrow sim(item_1, item_2)$

    $compatibility \leftarrow 0$
    for $uc \in profile$ **do**
        if $item_1$ satisfies uc in the respective feature dimension **then**
            $compatibility \leftarrow compatibility + 1$
    $compatibility \leftarrow compatibility / |profile|$

    return $similarity \cdot compatibility$

A.2 Concrete Feature Specifications

A.2.1 Flight Feature Specification

Feature	Name	Value Overview and Semantics	Example Value
A_1	Depature Country	Country of departure indicated by three letter codes	GER
A_2	Departure Airport	Airport of departure indicated by three letter codes	FRA
A_3	Arrival Country	Country of arrival indicated by three letter codes	GER
A_4	Arrival Airport	Airport of arrival indicated by three letter codes	FRA
A_5	Departure Date and Time	Date and time of day of the departure in datetime-format	10.10.17 - 15:30
A_6	Arrival Date and Time	Date and time of day of the departure in datetime-format	10.10.17 - 15:30
A_7	Carrier	Airline carrying the flight given by a two letter and digit code	CL
A_8	Flight Number	Flight number of the flight given by a three letter and three digit combination	DLH456
A_{18}	Fares per Traveler	A mapping of all types of available traveler types encoded by three letters to a fare specified via a price value and a currency	['ADT' $\mapsto$ 300$]
A_9	Change of Airport	Boolean value indicating whether a change of aiport is necessary during this connection	TRUE
A_{10}	Number of Stops	Natural number specifying the number of stops during the flight	5
A_{11}	Flight Duration	Natural number specifying the flight duration rounded to hours	5
A_{12}	Available Meals	List of two letter codes indicating the availability of meals	['V', 'BR']
A_{13}	Flight Frequency per Week	Natural number below eight indicating how often flights with the given flight number take place per week	5
A_{14}	eTicketing Availability	Boolean value indicating whether eTicketing is available	TRUE
A_{15}	Miles	Natural number indicating the number of miles flown during the flight	200
A_{16}	Airplane Code	Character string indicating the type of airplane of the flight	A380
A_{17}	Class of Service	The classes of service available for this flight specified by a one letter code	J

Table A.1.: Specification of the features of flights as given by the domain. The concrete model in the implemented BFRS maps the values of the complex feature domains described by lists or maps onto binary vectors. The values above the double horizontal line are mandatory the others are optional.

A.2.2 Sub-query feature Specification

Feature	Name	Value Overview and Semantics	Example Value
Q_1	Depature Cities	List of possible departure cities represented by three letter codes	FRA
Q_2	Arrival Cities	List of possible arrival cities represented by three letter codes	FRA
Q_3	Departure Date and Time	Preferred date and time of the departure of the flight	16.10.2017 - 9:50
Q_4	Traveler Types	List of traveler types as three letter codes associated with a number of travelers of that type	[ADT $\mapsto$ 2]
Q_5	Arrival Date and Time	Preferred date and time of the arrival of the flight	16.10.2017 - 9:50
Q_6	Hour Window before Departure	Natural number indicating the hours before the specified departure time, which are considered valid alternatives	3
Q_7	Hour Window after Departure	Natural number indicating the hours after the specified departure time, which are considered valid alternatives	3
Q_8	Maximum Total Flight Duration	Natural number indicating the maximum number of hours the flight should take	3
Q_9	Day Window Before Departure	Natural number indicating the days before the specified departure time, which are considered valid alternatives	1
Q_{10}	Day Window After Departure	Natural number indicating the days after the specified departure time, which are considered valid alternatives	1
Q_{11}	Airline Preference	Three letter code indicating the preferred air line as a carrier for the journey	CL
Q_{12}	Preference for Non-Stop and Direct Flights	Boolean value specifying the preference for non-stop, direct flights along the journey	TRUE
Q_{13}	Preference for Stop and Direct Flights	Boolean value specifying the preference for direct flights which in turn may have stops along the journey	TRUE

Table A.2.: Specification of the features of sub-queries as given by the underlying flight retrieval engine of the BFRS.

A.3 Evaluation Configurations

A.3.1 Evaluation Configurations of the Offsetting Components

Acronym	Sub-query Weights	Local Weights	Query
UUF	Uniform	Uniform	Full
UUR			Reduced
U1F	Uniform	$Local_1$	Full
U1R			Reduced
U2F	Uniform	$Local_2$	Full
U2R			Reduced
U3F	Uniform	$Local_3$	Full
U3R			Reduced
1UF	Sub_1	Uniform	Full
1UR			Reduced
11F	Sub_1	$Local_1$	Full
11R			Reduced
12F	Sub_1	$Local_2$	Full
12R			Reduced

Table A.3.: The configurations considered during the development phase of the query-based RS. The denominators *Uniform* specify the weight configuration of equal weights, $Local_{1,\,2,\,3}$ and Sub_1 are non-uniform weights defined with domain expert knowledge. Here $Local_1$ and $Local_2$, intuitively speaking, emphasize locational and timely attributes and $Local_3$ emphasizes timely and comfort properties. Sub_1 puts higher weight on the sub-query specifying the first itinerary.

Acronym	Classification Approach	Sampling Approach
NBU1:1	Naïve Bayes	Under-sampling 1:1
NBU5:5	Naïve Bayes	Under- & Over-sampling 5:5
NBS5:5	Naïve Bayes	SMOTE 5:5
NBS3:3	Naïve Bayes	SMOTE 3:3
3NN1:1	3-Nearest-Neighbor	Under-sampling 1:1
3NN1:3	3-Nearest-Neighbor	Under-sampling 1:3
3NN1:2	3-Nearest-Neighbor	Under-sampling 1:2
5NN1:3	5-Nearest-Neighbor	Under-sampling 1:3
10NN1:3	10-Nearest-Neighbor	Under-sampling 1:3

Table A.4.: The development configurations for the long-term-preference-based RS. The right most column defines the sampling approach, where entries of the form *under- (& over-)sampling x:y* refer to random under-sampling having y items chosen from the majority class and x items from the minority class. *SMOTE x:y* refers to a SMOTE variant where x synthetic minority items and y negative samples are added to the training data.

Acronym	Parallel Hybrid Approach	Parameters
M0.5/0.5	Mixed	$n_{long} = 0.5$, $n_{short} = 0.5$, $\tau = 0$
M0.3/0.7	Mixed	$n_{long} = 0.3$, $n_{short} = 0.7$, $\tau = 0$
M0.7/0.3	Mixed	$n_{long} = 0.7$, $n_{short} = 0.3$, $\tau = 0$
M0.5/0.5-0.2	Mixed	$n_{long} = 0.5$, $n_{short} = 0.5$, $\tau = 0.2$
W0.5/0.5	Weighted	$\alpha_{long} = 0.5$, $\alpha_{short} = 0.5$
W0.3/0.7	Weighted	$\alpha_{long} = 0.3$, $\alpha_{short} = 0.7$
S0.1	Switching	$certainty(rec_{long}) < 0.1$
S0.3	Switching	$certainty(rec_{long}) < 0.3$
S0.5	Switching	$certainty(rec_{long}) < 0.5$

Table A.5.: The development configurations for the whole offsetting module. Here the parameters refer to the respective parameters in the definitions of the different parallel hybridization schemes 2.3.6 and the formula of the implemented, mixed hybrid 3.3.6.

A.3.2 Evaluation Configurations of the Conversational Components

Acronym	Critiquing Approach	Presented Options	Critique Option Ranking	Support Threshold
D15/8-max-0.1	Dynamic	7:8	Maximum Support	0.1
D15/8-min-0.1	Dynamic	7:8	Minimal Support	0.1
D15/8-ran-0.1	Dynamic	7:8	Random	0.1
D15/5-ran-0.1	Dynamic	10:5	Random	0.1
D25/5-ran-0.1	Dynamic	20:5	Random	0.1
D25/10-ran-0.1	Dynamic	15:10	Random	0.1
D25/10-max-0.1	Dynamic	15:10	Maximum Support	0.1
D25/10-min-0.1	Dynamic	15:10	Minimum Support	0.1
D25/10-ran-0.2	Dynamic	15:10	Random	0.2
D25/10-ran-0.3	Dynamic	15:10	Random	0.3
D25/10-ran-0.4	Dynamic	15:10	Random	0.4
D25/5-ran-0.14	Dynamic	20:5	Random	0.14
D25/10-ran-0.14	Dynamic	15:10	Random	0.14
D15/0-ran	Dynamic Unit	15:0	Random	-
D20/0-ran	Dynamic Unit	20:0	Random	-
S20-ran	Static Unit	20:0	Random	-

Table A.6.: The development configurations for the evaluation of the conversational module of the BFRS. The presented options with entries in the format *x:y* refer to the number of unit critiques (x) and the number of compound critiques (y) presented. That means in total $x + y$ critiques are presented. The support threshold is the lower support boundary for the Apriori algorithm underlying dynamic critiquing.

Acronym	Designator	Offsetting	Longterm-based Configuration	Query-based Configuration	Critiquing Configuration
SU/nO	Static Unit Baseline	None	-	-	S20-ran
D/nO	Dynamic Baseline	None	-	-	D25/5-ran-0.14
SU/nL	Static Unit with Query-based Offset	Query-based	-	12F	S20-ran
D/nL	Dynamic with Query-based Offset	Query-based	-	12F	D25/5-ran-0.14
SU/nQ	Static Unit with Long-term-based Offset	Long-term-based	NBU1:1	-	S20-ran
D/nQ	Dynamic with Long-term-based Offset	Long-term-based	NBU1:1	-	D25/5-ran-0.14
SU/O	Static Unit with Offset	Complete	NBU1:1	12F	S20-ran
D/O	Dynamic with Offset	Complete	NBU1:1	12F	D25/5-ran-0.14

Table A.7.: The configurations for the evaluation on the test dataset of the complete BFRS. The scenarios differing in the change of one component (i.e. the complete offsetting module, the long-term-preference-based RS and the query-based RS) are split by an empty row in the table. For each scenario a static unit critiquing and the dynamic critiquing approach are compared.

A.4 Evaluation Results

A.4.1 Evaluation Results of the Offsetting Components

	Theoretical Baseline				Practical Baseline			
k	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
1	0.006448	0.006448	0.006448	0.989375	0.017881	0.0178808	0.017881	0.994766
3	0.006448	0.006448	0.006448	0.975126	0.016555	0.049668	0.024834	0.994935
5	0.006448	0.006448	0.006448	0.971841	0.015497	0.077483	0.025828	0.995083
10	0.006448	0.006448	0.006448	0.964260	0.012848	0.1284768	0.023359	0.995355

Table A.8.: Evaluation results by the top-k-metrics for $k \in \{1, 3, 5, 10\}$ both for the theoretical and practical baselines.

	Uniform Weights Configuration (UUF)				Best Configuration (12F)			
k	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
1	0.039647	0.039647	0.039647	0.986834	0.041850	0.041850	0.041850	0.986855
3	0.029368	0.088105	0.044052	0.975553	0.035242	0.105726	0.052863	0.975933
5	0.026872	0.134361	0.044787	0.964114	0.029955	0.149779	0.049926	0.964549
10	0.025110	0.251101	0.045654	0.935768	0.028193	0.281938	0.051261	0.936184

Table A.9.: Evaluation results by the top-k-metrics for $k \in \{1, 3, 5, 10\}$ showed for a selected set of configurations considered during the development phase.

	Best Configuration Full Query (12F)				Best Configuration Reduced Query (12R)			
k	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
1	0.066162	0.066162	0.066162	0.986796	0.066162	0.0661625	0.066162	0.986796
3	0.0441083	0.132325	0.066162	0.975126	0.044108	0.132325	0.066162	0.975126
5	0.035396	0.176748	0.058979	0.963277	0.035396	0.176748	0.058979	0.963277
10	0.029766	0.295841	0.054059	0.9333079	0.029766	0.295841	0.054059	0.933307

Table A.10.: Evaluation results by the top-k-metrics for $k \in \{1, 3, 5, 10\}$ showed for the best configuration with full queries (left) and reduced queries (right) of the query-based recommender system on test data.

Trained on History of at most 5 most recent Feedbacks								
	Naïve Bayes with SMOTE 3:3 (NBS3:3)				Naïve Bayes with Under-sampling 1:1 (NBU1:1)			
k	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
1	0.058441	0.0584415	0.0584415	0.9898898	0.1103896	0.1103896	0.1103896	0.9904476
3	0.045454	0.1363636	0.0681818	0.9799888	0.0887445	0.2662337	0.1331168	0.9813833
5	0.050649	0.2532467	0.0844155	0.9705062	0.0870129	0.4350649	0.1450216	0.9724585
10	0.037012	0.3701298	0.0672963	0.9449170	0.0590909	0.5909090	0.1074380	0.9472876
Trained on History of at most 10 most recent Feedbacks								
	Naïve Bayes with SMOTE 3:3 (NBS3:3)				Naïve Bayes with Under-sampling 1:1 (NBU1:1)			
k	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
1	0.051948	0.0519480	0.0519480	0.9898201	0.1363636	0.1363636	0.1363636	0.9907265
3	0.045454	0.1363636	0.0681818	0.9799888	0.1017316	0.3051948	0.1525974	0.9818017
5	0.046753	0.2337662	0.0779220	0.9702970	0.0922077	0.4610389	0.1536796	0.9727374
10	0.037012	0.3701298	0.0672963	0.9449170	0.0623376	0.6233766	0.1133412	0.9476363
Trained on History of at most 15 most recent Feedbacks								
	Naïve Bayes with SMOTE 3:3 (NBS3:3)				Naïve Bayes with Under-sampling 1:1 (NBU1:1)			
k	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
1	0.058441	0.0584415	0.0584415	0.9898898	0.1168831	0.1168831	0.1168831	0.9905173
3	0.047619	0.1428571	0.0714285	0.9800585	0.0974025	0.2922077	0.1461038	0.9816622
5	0.048051	0.2402597	0.0800865	0.9703667	0.0935064	0.4675324	0.1558441	0.9728071
10	0.039610	0.3961038	0.0720188	0.9451959	0.0616883	0.6168831	0.1121605	0.9475665
Trained on History of all Feedbacks								
	Naïve Bayes with SMOTE 3:3 (NBS3:3)				Naïve Bayes with Under-sampling 1:1 (NBU1:1)			
k	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
1	0.058441	0.0584415	0.0584415	0.9898898	0.1038961	0.1038961	0.1038961	0.9903779
3	0.049783	0.1493506	0.0746753	0.9801282	0.0930735	0.2792207	0.1396103	0.9815228
5	0.049350	0.2467532	0.0822510	0.9704364	0.0857142	0.4285714	0.1428571	0.9723887
10	0.037662	0.3766233	0.0684769	0.9449867	0.0577922	0.5779220	0.1050767	0.9471482

Table A.11.: Evaluation results on the development set by the top-k-metrics for $k \in \{1, 3, 5, 10\}$ showed for at most 5, 10, 15 or all feedback instances as the underlying training data. On the left the performance of the Naïve Bayes classifier using SMOTE and on the right using under-sampling can be found.

	3NN1:3 on at most 5 most recent Feedbacks				3NN1:3 on at most 10 most recent Feedbacks			
k	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
1	0.0758620	0.0714285	0.0735785	0.9903430	0.0855263	0.0844155	0.0849673	0.9902384
3	0.0298719	0.2727272	0.0538461	0.9485427	0.0328542	0.3116883	0.0594427	0.9470436
5	0.0627907	0.1753246	0.0924657	0.9815228	0.0626398	0.1818181	0.0931780	0.9809998
10	0.0476190	0.2207792	0.0783410	0.9721098	0.0459459	0.2207792	0.0760626	0.9712034
	3NN1:3 on all Feedbacks							
k	**Precision**	**Recall**	**F1-Score**	**Accuracy**				
1	0.0980392	0.0974026	0.0977198	0.9903430				
3	0.0326530	0.3116883	0.0591133	0.9467298				
5	0.0665188	0.1948051	0.0991735	0.9809998				
10	0.0482573	0.2337662	0.08	0.9711337				

Table A.12.: Evaluation results on the development set by the top-k-metrics for $k \in \{1, 3, 5, 10\}$ showed for at most 5, 10 or all feedback instances as the training data. The performance is measured for the best performing classifier using 3NN and under-sampling by selecting 3 random negative examples from each candidate set.

	NBU1:1 on at most 5 most recent Feedbacks				NBU1:1 on at most 10 most recent Feedbacks			
k	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
1	0.102389	0.102389	0.102389	0.990357	0.112627	0.112627	0.112627	0.990467
3	0.078498	0.235494	0.117747	0.981045	0.078498	0.235494	0.117747	0.981045
5	0.059385	0.296928	0.098976	0.970962	0.059385	0.296928	0.098976	0.970962
10	0.037201	0.372013	0.067638	0.944913	0.038907	0.389078	0.070741	0.945097
	NBU1:1 on at most 15 most recent Feedbacks				NBU1:1 on all Feedbacks			
k	**Precision**	**Recall**	**F1-Score**	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**Accuracy**
1	0.119453	0.119453	0.119453	0.990540	0.122866	0.122866	0.122866	0.990577
3	0.080773	0.242320	0.121160	0.981118	0.076222	0.228668	0.114334	0.980971
5	0.060068	0.300341	0.100113	0.970999	0.058703	0.293515	0.097838	0.970926
10	0.039931	0.399317	0.072603	0.945207	0.037542	0.375426	0.068259	0.944950

Table A.13.: Evaluation results on the test set by the top-k-metrics for $k \in \{1, 3, 5, 10\}$ showed for at most 5, 10, 15 or all feedback instances as the underlying training data. The system was evaluated in the best performing configuration (NBU1:1).

Trained on History of at most 5 most recent Feedbacks								
	Best Mixed Composition (M0.5/0.5)				Best Switching Composition (S0.1)			
k	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.129870	0.129870	0.129870	0.990656	0.110389	0.110389	0.110389	0.990447
3	0.093073	0.279220	0.139610	0.981522	0.084415	0.253246	0.126623	0.981243
5	0.088311	0.441558	0.147186	0.972528	0.081818	0.409090	0.136363	0.972179
10	0.060389	0.603896	0.109799	0.947427	0.055844	0.558441	0.101534	0.946939
Trained on History of at most 10 most recent Feedbacks								
	Best Mixed Composition (M0.5/0.5)				Best Switching Composition (S0.1)			
k	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.155844	0.155844	0.155844	0.990935	0.129870	0.129870	0.129870	0.990656
3	0.114718	0.344155	0.172077	0.982220	0.093073	0.279220	0.139610	0.981522
5	0.096103	0.480519	0.160173	0.972946	0.084415	0.422077	0.140692	0.972319
10	0.068181	0.681818	0.123966	0.948263	0.059740	0.597402	0.108618	0.947357
Trained on History of all Feedbacks								
	Best Mixed Composition (M0.5/0.5)				Best Switching Composition (S0.1)			
k	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.123376	0.123376	0.123376	0.990587	0.097402	0.097402	0.097402	0.990308
3	0.097402	0.292207	0.146103	0.981662	0.086580	0.259740	0.129870	0.981313
5	0.087012	0.435064	0.145021	0.972458	0.080519	0.402597	0.134199	0.972109
10	0.061688	0.616883	0.112160	0.947566	0.055844	0.558441	0.101534	0.946939

Table A.14.: Evaluation results on the development set by the top-k-metrics for $k \in \{1, 3, 5, 10\}$ showed for at most 5, 10, 15 or all feedback instances as the underlying training data. On the left the performance of the offsetting module using mixed composition and on the right switching composition in their best configurations are listed.

	Offsetting Module on at most 5 most recent Feedbacks				Offsetting Module on at most 10 most recent Feedbacks			
k	Precision	Recall	F1-Score	Accuracy	Precision	Recall	F1-Score	Accuracy
1	0.105802	0.105802	0.105802	0.990394	0.119453	0.119453	0.119453	0.990540
3	0.077360	0.232081	0.116040	0.981008	0.078498	0.235494	0.117747	0.981045
5	0.055290	0.276450	0.092150	0.970742	0.059385	0.296928	0.098976	0.970962
10	0.039631	0.395904	0.072049	0.945225	0.040655	0.406143	0.073913	0.945335
	Offsetting Module on all recent Feedbacks							
k	Precision	Recall	F1-Score	Accuracy				
1	0.133105	0.133105	0.133105	0.990687				
3	0.079635	0.238907	0.119453	0.981081				
5	0.059385	0.296928	0.098976	0.970962				
10	0.040997	0.409556	0.074534	0.945372				

Table A.15.: Evaluation results of the offsetting module on the test set by the top-k-metrics for $k \in \{1, 3, 5, 10\}$ showed for at most 5, 10, 15 or all feedback instances as the underlying training data.

A.4.2 Evaluation Results of the Conversational Components

Critiquing Constellation	Average Path Length	Proportion Aborted Sessions	Proportion Unit Critiques Selected	Average Critique Option Size
D15/8-max-0.1	76.34963219	0.173950671	0.906366893	2.035604637
D15/8-min-0.1	75.98485504	0.174816097	0.903428796	2.037224086
D15/8-ran-0.1	72.30942275	0.196519525	0.89509664	2.119955156
D25/10-max-0.1	76.01696390	0.170508917	0.900562476	2.194823632
D25/10-min-0.1	76.62007789	0.179575941	0.901287069	2.186120794
D25/10-ran-0.1	72.28862479	0.199490662	0.884990013	2.277745218
D25/5-ran-0.1	72.93194450	0.172723867	0.90951941	1.725908119
D25/10-ran-0.2	80.73441296	0.189473684	0.846485202	2.362135838
D25/10-ran-0.3	80.58316553	0.189689891	0.774214346	2.365733624
D25/10-ran-0.4	81.44502618	0.192509062	0.800912831	2.204423033
D15/0-ran	81.3753230	0.204134367	1	1

Table A.16.: Evaluation results of the conversational RS on the development set by the typical metrics used for critiquing-based RS. The different experiments focusing on one parameter are separated by an line in the table.

Critiquing Constellation	Average Path Length	Proportion Aborted Sessions	Proportion Unit Critiques Selected	Average Critique Option Size
D25/5-ran-0.14 (Dynamic)	76.43906075	0.183376817	0.902757867	1.788246798
D20/0-ran (Dynamic Unit)	78.18449497	0.185240403	1	1
S20-ran (Static Unit)	82.07418509	0.189584114	1	1

Table A.17.: Evaluation results of the conversational RS on the test set by the typical metrics used for critiquing-based RS. Static and dynamic unit critiquing are compared with the best configuration of a dynamic critiquing approach.

A.4.3 Evaluation Results of the Complete Business Flight Recommender System

System Configuration	Average Path Length	Proportion Aborted Sessions	Proportion Unit Critiques Selected	Average Critique Option Size
SU/nO	75.04026846	0.185682327	1	1
D/nO	74.13422819	0.185682327	0.897247873	1.794205723

Table A.18.: Evaluation results of the baselines for the complete BFRS on the test set by the typical metrics used for critiquing-based RS. Static unit critiquing is compared with the best configuration of a dynamic critiquing approach.

System Configuration	Average Path Length	Proportion Aborted Sessions	Proportion Unit Critiques Selected	Average Critique Option Size
SU/nL	73.79418345	0.1901566	1	1
D/nL	68.52572707	0.172259508	0.894388038	1.805289813

Table A.19.: Evaluation results of the BFRS using only the query-based RS as an offset on the test set by the typical metrics used for critiquing-based RS. Static unit critiquing is compared with the best configuration of a dynamic critiquing approach.

Trained on History of at most 5 most recent Feedbacks				
	Average Path Length	Proportion Aborted Sessions	Proportion Unit Critiques Selected	Average Critique Option Size
SU/nQ	75.63087248	0.1901566	1	1
DU/nQ	72.80536913	0.194630872	0.90028884	1.815083073
Trained on History of at most 10 most recent Feedbacks				
	Average Path Length	Proportion Aborted Sessions	Proportion Unit Critiques Selected	Average Critique Option Size
SU/nQ	74.81655481	0.1901566	1	1
DU/nQ	68.75838926	0.172259508	0.866320446	1.81745783
Trained on History of all Feedbacks				
	Average Path Length	Proportion Aborted Sessions	Proportion Unit Critiques Selected	Average Critique Option Size
SU/nQ	75.2147651	0.192393736	1	1
DU/nQ	71.37136465	0.18344519	0.896216657	1.80677388

Table A.20.: Evaluation results on the test set by the usual critiquing metrics showed for at most 5, 10 or all feedback instances as the underlying training data. The BFRS with the query-based RS turned off is evaluated.

Trained on History of at most 5 most recent Feedbacks				
	Average Path Length	Proportion Aborted Sessions	Proportion Unit Critiques Selected	Average Critique Option Size
SU/O	75.7114094	0.196868009	1	1
DU/O	69.93288591	0.181208054	0.896641075	1.808100591
Trained on History of at most 10 most recent Feedbacks				
	Average Path Length	Proportion Aborted Sessions	Proportion Unit Critiques Selected	Average Critique Option Size
SU/O	73.73825503	0.181208054	1	1
DU/O	65.54362416	0.163310962	0.896409311	1.814935387
Trained on History of all Feedbacks				
	Average Path Length	Proportion Aborted Sessions	Proportion Unit Critiques Selected	Average Critique Option Size
SU/O	71.02684564	0.178970917	1	1
DU/O	64.46756152	0.158836689	0.889926085	1.80677388

Table A.21.: Evaluation results on the test set by the usual critiquing metrics showed for at most 5, 10 or all feedback instances as the underlying training data. The complete Business Flight Recommender System in its final configuration is evaluated.

List of Figures

List of Tables

List of Algorithms

Acronyms

BFRS Business Flight Recommender System

CBF Content-based Filtering

CBRS Critiquing-based Recommender System

CF Collaborative Filtering

ConvRS Conversational Recommender System

DR Demographic Recommendation

HRS Hybrid Recommender System

IF Information Filtering

IR Information Retrieval

KBR Knowledge-based Recommendation

MAUT Multi-attribute Utility Theory

RS Recommender System

VSM Vector Space Model

XML Extensible Markup Language

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Institute of Electrical and Electronics Engineers (IEEE) transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[3] Klaus-Dieter Althoff and Michael M Richter. Similarity and utility in non-numerical domains. In *Mathematische Methoden der Wirtschaftswissenschaften*, pages 403–413. Springer, 1999.

[4] Sarabjot Singh Anand and Bamshad Mobasher. Contextual recommendation. In *Workshop on Web Mining*, pages 142–160. Springer, 2006.

[5] VDR The German Business Travel Association. Vdr business travel report 2017 - volume 15, 2017. Online available at https://www.vdr-service.de/der-verband/fachmedien/vdr-geschaeftsreiseanalyse/ (25.10.2017).

[6] Raul Sérgio Barth. Design and implementation of a flight recommendation engine, June 2014. Graduation Project, Universidade Federal do Rio Rande do Sul.

[7] Joeran Beel, Stefan Langer, Marcel Genzmehr, Bela Gipp, Corinna Breitinger, and Andreas Nürnberger. Research paper recommender system evaluation: A quantitative literature survey. In *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation*, Reproducibility and Replication in Recommender Systems Evaluation 2013 (RepSys '13), pages 15–22. Association of Computing Machinery (ACM), 2013.

[8] Nicholas J Belkin and W Bruce Croft. Information filtering and information retrieval: Two sides of the same coin? *Communications of the Association of Computing Machinery*, 35(12):29–38, 1992.

[9] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.

[10] Shyam Boriah, Varun Chandola, and Vipin Kumar. Similarity measures for categorical data: A comparative evaluation. In *Proceedings of the 2008 Society for Industrial and Applied Mathematics (SIAM) International Conference on Data Mining*, pages 243–254. Society for Industrial and Applied Mathematics (SIAM), 2008.

[11] Derek Bridge, Mehmet H Göker, Lorraine McGinty, and Barry Smyth. Case-based recommender systems. *The Knowledge Engineering Review*, 20(3):315–320, 2005.

[12] Robin Burke. The wasabi personal shopper: a case-based recommender system. In *Association for the Advancement of Artificial Intelligence / Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI)*, pages 844–849, 1999.

[13] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[14] Robin D Burke, Kristian J Hammond, and BC Yound. The findme approach to assisted browsing. *Institute of Electrical and Electronics Engineers (IEEE) Expert*, 12(4):32–40, 1997.

[15] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority oversampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[16] Li Chen and Pearl Pu. Critiquing-based recommenders: survey and emerging trends. *User Modeling and User-Adapted Interaction*, 22(1):125–150, 2012.

[17] D Manning Christopher, Raghavan Prabhakar, and SCHÜTZE Hinrich. Introduction to information retrieval. *An Introduction To Information Retrieval*, 151, 2008.

[18] Amazon cited from de.statista.com. Anzahl aktiver kunden-accounts von amazon weltweit in den jahren 1997 bis 2015 (in millionen), January 2016. Accessible online via https://de.statista.com/statistik/daten/studie/297615/umfrage/anzahl-weltweit-aktiver-kunden-accounts-von-amazon/.

[19] Marketplace Analytics cited from de.statista.com. Anzahl der gelisteten produkte bei amazon.de nach hauptkategorien im jahr 2016 (in millionen), 2017. Accessible online via https://de.statista.com/statistik/daten/studie/666152/umfrage/anzahl-der-gelisteten-produkte-bei-amazon-de-nach-kategorien/.

[20] Marco De Gemmis, Leo Iaquinta, Pasquale Lops, Cataldo Musto, Fedelucio Narducci, and Giovanni Semeraro. Preference learning in recommender systems. *Preference Learning*, 41, 2009.

[21] Paul Dourish. What we talk about when we talk about context. *Personal and ubiquitous computing*, 8(1):19–30, 2004.

[22] Jon Imanol Durán, Juhani Laitakari, Daniel Pakkala, and Juho Perälä. A user meta-model for context-aware recommender systems. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, pages 63–66. Association of Computing Machinery (ACM), 2010.

[23] Alexander Felfernig and Robin Burke. Constraint-based recommender systems: technologies and research issues. In *Proceedings of the 10th international conference on Electronic commerce*, page 3. Association of Computing Machinery (ACM), 2008.

[24] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Developing constraint-based recommenders. In *Recommender systems handbook*, pages 187–215. Springer, 2011.

[25] Salvador García, Sergio Ramírez-Gallego, Julián Luengo, José Manuel Benítez, and Francisco Herrera. Big data preprocessing: methods and prospects. *Big Data Analytics*, 1(1):9, 2016.

[26] Mehmet H Göker and Cynthia A Thompson. Personalized conversational case-based recommendation. In *European Workshop on Advances in Case-Based Reasoning*, pages 99–111. Springer, 2000.

[27] A Ardeshir Goshtasby. Similarity and dissimilarity measures. In *Image registration*, pages 7–66. Springer, 2012.

[28] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *Association of Computing Machinery (ACM) Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[29] Zan Huang, Hsiu-chin Chen, and Daniel Dajun Zeng. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *Association of Computing Machinery (ACM) Transactions on Information Systems*, 22:116–142, 01 2004.

[30] FO Isinkaye, YO Folajimi, and BA Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261–273, 2015.

[31] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender systems: an introduction*. Cambridge University Press, New York City, USA, 2010.

[32] Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim, and Sung Hyon Myaeng. Some effective techniques for naive bayes text classification. *Institute of Electrical and Electronics Engineers (IEEE) transactions on knowledge and data engineering*, 18(11):1457–1466, 2006.

[33] Fabiana Lorenzi, Francesco Ricci, R Tostes, and Rs Brasil. Case-based recommender systems: A unifying view. *Lecture notes in computer science*, 3169:89, 2005.

[34] Lorraine McGinty and James Reilly. On the evolution of critiquing recommenders. In *Recommender Systems Handbook*, pages 419–453. Springer, 2011.

[35] Lorraine McGinty and Barry Smyth. Tweaking critiquing. In *Proceedings of the Workshop on Personalization and Web Techniques at the International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 20–27, 2003.

[36] David McSherry and David W Aha. Avoiding long and fruitless dialogues in critiquing. *Research and Development in Intelligent Systems XXIII*, pages 173–186, 2007.

[37] David McSherry and David W Aha. The ins and outs of critiquing. In *IJCAI*, pages 962–967, 2007.

[38] Douglas W Oard, Jinmook Kim, et al. Implicit feedback for recommender systems. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI) workshop on recommender systems*, pages 81–83, Menlo Park, CA, 1998. Association for the Advancement of Artificial Intelligence (AAAI) Press.

[39] Kenta Oku, Shinsuke Nakajima, Jun Miyazaki, and Shunsuke Uemura. Context-aware svm for context-dependent information recommendation. In *Proceedings of the 7th International Conference on Mobile Data Management*, Mobile Data Management 2006 (MDM'06), pages 109–, Washington, DC, USA, 2006. Institute of Electical and Electronics Engineers (IEEE) Computer Society.

[40] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[41] Jérome Picault, Myriam Ribiere, David Bonnefoy, and Kevin Mercer. How to get the recommender out of the lab? In *Recommender Systems Handbook*, pages 333–365. Springer, 2011.

[42] Pearl Pu and Boi Faltings. Enriching buyers' experiences: the smartclient approach. In *Proceedings of the Special Interest Group on Computer-Human Interaction (SIGCHI) conference on Human Factors in Computing Systems*, pages 289–296. Association of Computing Machinery (ACM), 2000.

[43] Rachael Rafter and Barry Smyth. Conversational collaborative recommendation–an experimental analysis. *Artificial Intelligence Review*, 24(3-4):301–318, 2005.

[44] Bhavani Raskutti, Anthony Beitz, and Belinda Ward. A feature-based approach to recommending selections based on past preferences. *User Modeling and User-Adapted Interaction*, 7(3):179–218, 1997.

[45] James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. Dynamic critiquing. In *European Conference on Case-based Reasoning (ECCBR)*, volume 3155, pages 763–777. Springer, 2004.

[46] James Reilly, Kevin McCarthy, Lorraine McGinty, and Barry Smyth. Incremental critiquing. *Knowledge-Based Systems*, 18(4):143–151, 2005.

[47] James Reilly, Jiyong Zhang, Lorraine McGinty, Pearl Pu, and Barry Smyth. A comparison of two compound critiquing systems. In *Proceedings of the 12th international conference on Intelligent user interfaces*, pages 317–320. Association of Computing Machinery (ACM), 2007.

[48] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.

[49] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B Kantor. *Recommender systems handbook*. Springer, 2015.

[50] Yasser Salem, Jun Hong, and Weiru Liu. History-guided conversational recommendation. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 999–1004. Association of Computing Machinery (ACM), 2014.

[51] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[52] Klaus D. Schmidt. *Maß und Wahrscheinlichkeit*. Springer-Verlag Berlin Heidelberg, 2 edition, 2011.

[53] Brendon Towle and Clark Quinn. Knowledge based recommender systems using explicit user models. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI) Workshop on Knowledge-Based Electronic Markets*, pages 74–77, 2000.

[54] Shari Trewin. Knowledge-based recommender systems. *Encyclopedia of library and information science*, 69(Supplement 32):180, 2000.

[55] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[56] Jiyong Zhang and Pearl Pu. A comparative study of compound critique generation in conversational recommender systems. In *Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 234–243. Springer, 2006.

[57] Ingrid Zukerman and David W Albrecht. Predictive statistical models for user modeling. *User Modeling and User-Adapted Interaction*, 11(1-2):5–18, 2001.