# Basic Approaches in Recommendation Systems

Alexander Felfernig, Michael Jeran, Gerald Ninaus, Florian Reinfrank, Stefan Reiterer, and Martin Stettinger

**Abstract** Recommendation systems support users in finding items of interest. In this chapter, we introduce the basic approaches of collaborative filtering, content-based filtering, and knowledge-based recommendation. We first discuss principles of the underlying algorithms based on a running example. Thereafter, we provide an overview of hybrid recommendation approaches which combine basic variants. We conclude this chapter with a discussion of newer algorithmic trends, especially critiquing-based and group recommendation.

## 1 Introduction

Recommendation systems [6, 32] provide suggestions for items that are of potential interest for a user. These systems are applied for answering questions such as *which book to buy*? [38], *which web site to visit next*? [47], and *which financial service to choose*? [19]. In software engineering scenarios, typical questions that can be answered with the support of recommendation systems are, for example, *which software changes probably introduce a bug*? [2], *which requirements to implement in the next software release*? [24], *which stakeholders should participate in the upcoming software project*? [37], *which method calls might be useful in the current development context*? [57], *which software components (or APIs) to reuse?* [43], *which software artifacts are needed next*? [52], and *which effort estimation methods should be applied in the current project phase*? [48]. An overview of the applica-

Corresponding author: Alexander Felfernig
Institute for Software Technology, Inffeldgasse 16b/2, A-8010 Graz
e-mail: alexander.felfernig@ist.tugraz.at
Preprint Version, published by springer.com: A. Felfernig, M. Jeran, G. Ninaus, F. Reinfrank, and S. Reiterer, Basic Approaches in Recommendation Systems, Recommendation Systems in Software Engineering, Springer Verlag, pp. 15-37 Springer, 2014.

tion of different types of recommendation technologies in the software engineering context can be found in Robillard et al. [51].

The major goal of this book chapter is to shed light on the basic properties of the three major recommendation approaches of (a) **collaborative filtering** [11, 25, 35], (b) **content-based filtering** [47], and (c) **knowledge-based recommendation** [4, 15]. Starting with the basic algorithmic approaches, we exemplify the functioning of the algorithms and discuss criteria that help to decide which algorithm should be applied in which context.

The remainder of this chapter is organized as follows. In Section 2 we give an overview of collaborative filtering recommendation approaches. In Section 3 we introduce the basic concepts of content-based filtering. We close our discussion of basic recommendation approaches with the topic of knowledge-based recommendation (see Section 4). In Section 5, we explain example scenarios for integrating the basic recommendation algorithms into hybrid ones. Hints for practitioners interested in the development of recommender applications are given in Section 6. A short overview of further algorithmic approaches is presented in Section 7. This chapter is concluded with Section **??**.

## 2 Collaborative Filtering

The itemset in our running examples is *software engineering related learning material* offered, for example, on an e-learning platform (see Table 1). Each learning unit is additionally assigned to a set of categories, for example, the learning unit $l_1$ is characterized by *Java* and *UML*.

| learning unit (LU) | name | Java | UML | Management | Quality |
|---|---|---|---|---|---|
| $l_1$ | Data Structures in Java | yes | yes | | |
| $l_2$ | Object Relational Mapping | yes | yes | | |
| $l_3$ | Software Architectures | | yes | | |
| $l_4$ | Project Management | | yes | yes | |
| $l_5$ | Agile Processes | | | yes | |
| $l_6$ | Object Oriented Analysis | | yes | yes | |
| $l_7$ | Object Oriented Design | yes | yes | | |
| $l_8$ | UML and the UP | | yes | yes | |
| $l_9$ | Class Diagrams | | yes | | |
| $l_{10}$ | OO Complexity Metrics | | | | yes |

Table 1: Example set of software engineering related learning units (LU) – this set will be exploited for demonstration purposes throughout this chapter. Each of the learning units is additionally characterized by a set of categories (*Java*, *UML*, *Management*, *Quality*), for example, the learning unit $l_1$ is assigned to the categories *Java* and *UML*.

*Collaborative filtering* [11, 35, 55] is based on the idea of word-of-mouth promotion: the opinion of family members and friends plays a major role in personal decision making. In online scenarios (e.g., online purchasing [38]), family members and friends are replaced by so-called *nearest neighbors* who are users with a similar preference pattern or purchasing behavior compared to the current user. Collaborative filtering (see Figure 1) relies on two different types of *background data*: (a) a set of users and (b) a set of items. The relationship between users and items is primarily expressed in terms of *ratings* which are provided by users and exploited in future recommendation sessions for predicting the rating a user (in our case user $U_a$) would provide for a specific item. If we assume that user $U_a$ currently interacts with a collaborative filtering recommendation system, the first step of the recommendation system is to identify the *nearest neighbors* (users with a similar rating behavior compared to $U_a$) and to extrapolate from the ratings of the similar users the rating of user $U_a$.

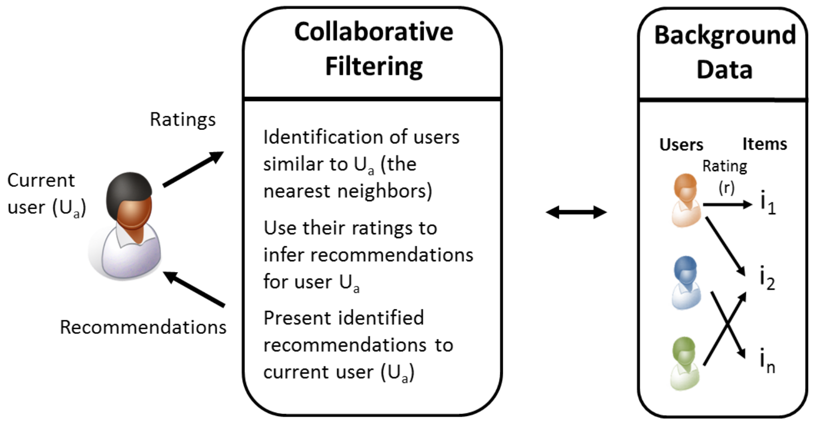
Ratings
Current user ($U_a$)
Recommendations
Collaborative Filtering
Identification of users similar to $U_a$ (the nearest neighbors)
Use their ratings to infer recommendations for user $U_a$
Present identified recommendations to current user ($U_a$)
Background Data
Users
Items
Rating (r)
$i_1$
$i_2$
$i_n$

Fig. 1: Collaborative filtering (CF) dataflow: users are rating items and receive recommendations for items based on the ratings of users with a similar rating behavior – the nearest neighbors (NN).

The basic procedure of collaborative filtering can best be explained based on a running example (see Table 2) which is taken from the software engineering domain (collaborative recommendation of learning units). Note that in this chapter we focus on so-called memory-based approaches to collaborative filtering which – in contrast to model-based approaches – operate on uncompressed versions of the user/item matrix [3]. The two basic approaches to collaborative filtering are *user-based collaborative filtering* [35] and *item-based collaborative filtering* [53]. Both variants are predicting to which extent the active user would be interested in items which have not been rated by her/him up to now.

| LU | name | $U_1$ | $U_2$ | $U_3$ | $U_4$ | $U_a$ |
|---|---|---|---|---|---|---|
| $l_1$ | Data Structures in Java | 5.0 | | | 4.0 | |
| $l_2$ | Object Relational Mapping | 4.0 | | | | |
| $l_3$ | Software Architectures | | 3.0 | 4.0 | 3.0 | |
| $l_4$ | Project Management | | 5.0 | 5.0 | | 4.0 |
| $l_5$ | Agile Processes | | | 3.0 | | |
| $l_6$ | Object Oriented Analysis | | 4.5 | 4.0 | | 4.0 |
| $l_7$ | Object Oriented Design | 4.0 | | | | |
| $l_8$ | UML and the UP | | 2.0 | | | |
| $l_9$ | Class Diagrams | | | | 3.0 | |
| $l_{10}$ | OO Complexity Metrics | | | | 5.0 | 3.0 |
| | average rating ($\overline{r_\alpha}$) | 4.33 | 3.625 | 4.0 | 3.75 | 3.67 |

Table 2: Example collaborative filtering data structure (rating matrix): learning units (LU) and related user ratings (we assume a rating scale of 1–5).

*User-based Collaborative Filtering*. User-based collaborative filtering identifies the k-nearest neighbors of the active user (see Formula 1)[1] and – based on these nearest neighbors – calculates a prediction of the active user's rating for a specific item (learning unit). In the example of Table 2, user $U_2$ is the nearest neighbor (k=1) of user $U_a$ (based on Formula 1) and his/her rating of learning unit $l3$ will be taken as a prediction for the rating of $U_a$ (rating = 3.0). The similarity between a user $U_a$ (the current user) and another user $U_x$ can be determined, for example, based on the Pearson correlation coefficient [32] (see Formula 1) where $LU_c$ is the set of items that have been rated by both users, $r_{\alpha,l_i}$ is the rating of user $\alpha$ for item $l_i$, and $\overline{r_\alpha}$ is the average rating of user $\alpha$. Similarity values resulting from the application of Formula 1 can take values on a scale of [-1 .. +1].

$$similarity(U_a,U_x) = \frac{\sum_{l_i \in LU_c}(r_{a,l_i} - \overline{r_a}) \times (r_{x,l_i} - \overline{r_x})}{\sqrt{\sum_{l_i \in LU_c}(r_{a,l_i} - \overline{r_a})^2} \times \sqrt{\sum_{l_i \in LU_c}(r_{x,l_i} - \overline{r_x})^2}} \quad (1)$$

The similarity values for $U_a$ calculated based on Formula 1 are shown in Table 3. For the purposes of our example we assume the existence of at least two items per user pair ($U_i$, $U_j$) (i $\neq$ j) in order to be able to determine a similarity. This criterion holds for users $U_2$ and $U_3$.

| | $U_1$ | $U_2$ | $U_3$ | $U_4$ |
|---|---|---|---|---|
| $U_a$ | - | 0.97 | 0.70 | - |

Table 3: Similarity between user $U_a$ and the users $U_j \neq U_a$ determined based on Formula 1. If the number of commonly rated items is below 2, no similarity between the two users is calculated.

[1] For simplicity we assume k=1 throughout this paper.

A major challenge in the context of estimating the similarity between users is the *sparsity* of the rating matrix since users are typically providing ratings for only a very small subset of the set of offered items. For example, given a large movie dataset that contains thousands of entries, a user will typically be able to rate only a few dozens. A basic approach to tackle this problem is to take into account the number of commonly rated items in terms of a *correlation significance* [29], i.e., the higher the number of commonly rated items, the higher is the significance of the corresponding correlation. For further information regarding the handling of sparsity we refer the reader to [29, 32].

The information about the set of users with a similar rating behavior compared to the current user (nearest neighbors NN) is the basis for predicting the rating of user $U_a$ for an *item* that has not been rated up to now by $U_a$ (see Formula 2).

$$prediction(U_a, item) = \overline{r_a} + \frac{\sum_{U_j \in NN} similarity(U_a, U_j) \times (r_{j,item} - \overline{r_j})}{\sum_{U_j \in NN} similarity(U_a, U_j)} \tag{2}$$

Based on the rating of the nearest neighbor of $U_a$, we are able to determine a prediction for user $U_a$ (see Table 4). The nearest neighbor of $U_a$ is user $U_2$ (see Table 3). The learning units rated by $U_2$ but not rated by $U_a$ are $l3$ and $l8$. Due to the determined predictions (Formula 2), item $l3$ would be ranked higher than item $l8$ in a recommendation list.

| | $l_1$ | $l_2$ | $l_3$ | $l_4$ | $l_5$ | $l_6$ | $l_7$ | $l_8$ | $l_9$ | $l_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $U_2$ | - | - | 3.0 | 5.0 | - | 4.5 | - | 2.0 | - | - |
| $U_a$ | - | - | - | 4.0 | - | 4.0 | - | | - | 3.0 |
| $prediction\ for\ U_a$ | - | - | **3.045** | - | - | - | - | **2.045** | - | - |

Table 4: User-based collaborative filtering based recommendations (predictions) for items that have not been rated by user $U_a$ up to now.

*Item-based Collaborative Filtering*. In contrast to user-based collaborative filtering, item-based collaborative filtering searches for items (nearest neighbors – NN) rated by $U_a$ that received similar ratings as items currently under investigation. In our running example, learning unit $l4$ has already received a good evaluation (4.0 on a rating scale of 1–5) by $U_a$. The item which is most similar to $l4$ and has not been rated by $U_a$ is item $l3$ (similarity($l_3, l_4$) = 0.35). In this case, the nearest neighbor of item $l3$ is $l4$ (this calculation is based on Formula 3).

If we want to determine a recommendation based on item-based collaborative filtering, we have to determine the similarity (using the Pearson correlation coefficient) between two items $l_a$ and $l_b$ where U denotes the set of users who both rated $l_a$ and $l_b$, $r_{u,l_i}$ denotes the rating of user u on item $l_i$, and $\overline{r_{l_i}}$ is the average rating of the i-th item.

$$similarity(l_a, l_b) = \frac{\sum_{u \in U}(r_{u,l_a} - \overline{r_{l_a}}) \times (r_{u,l_b} - \overline{r_{l_b}})}{\sqrt{\sum_{u \in U}(r_{u,l_a} - \overline{r_{l_a}})^2} \times \sqrt{\sum_{u \in U}(r_{u,l_b} - \overline{r_{l_b}})^2}} \quad (3)$$

The information about the set of items with a similar rating pattern compared to the *item* under consideration (nearest neighbors NN) is the basis for predicting the rating of user $U_a$ for the *item* (see Formula 4). Note that in this case NN represents a set of items already evaluated by $U_a$. Based on the assumption of k=1, $prediction(U_a, l_3)$ = 4.0, i.e., user $U_a$ would rate item $l_3$ with 4.0.

$$prediction(U_a, item) = \frac{\sum_{it \in NN} similarity(item, it) \times r_{a,it}}{\sum_{it \in NN} similarity(item, it)} \quad (4)$$

For a discussion of advanced collaborative recommendation approaches we refer the reader to [36, 53].

## 3 Content-based Filtering

*Content-based filtering* [47] is based on the assumption of monotonic personal interests. For example, users interested in the topic *Operating Systems* are typically not changing their interest profile from one day to another but will also be interested in the topic in the (near) future. In online scenarios, content-based recommendation approaches are applied, for example, when it comes to the recommendation of websites [47] (news items with a similar content compared to the set of already consumed news).

Content-based filtering (see Figure 2) relies on two different types of background data: (a) a set of users and (b) a set of categories (or keywords) that have been assigned to (or extracted from) the available items (item descriptions). Content-based filtering recommendation systems calculate a set of items that are most similar to items already known to the current user ($U_a$).

The basic approach of content-based filtering is to compare the content of already consumed items (e.g., a list of news articles) with new items that can potentially be recommended to the user, i.e., to find items that are similar to those already consumed (positively rated) by the user. The basis for determining such a similarity are *keywords* extracted from the item content descriptions (e.g., keywords extracted from news articles) or *categories* in the case that items have been annotated with the relevant meta-information. Readers interested in the principles of keyword extraction are referred to Jannach et al. [32]. Within the scope of this chapter we focus on content-based recommendation which exploits item categories (see Table 1).

Content-based filtering will now be explained based on a running example which relies on the information depicted in Tables 1, 5, and 6. Table 1 provides an overview of the relevant items and the assignments of items to categories. Table 5 provides information on which categories are of relevance for the different users. For example, user $U_1$ is primarily interested in items related to the categories *Java* and *UML*. In our running example, this information has been derived from the rating matrix de-

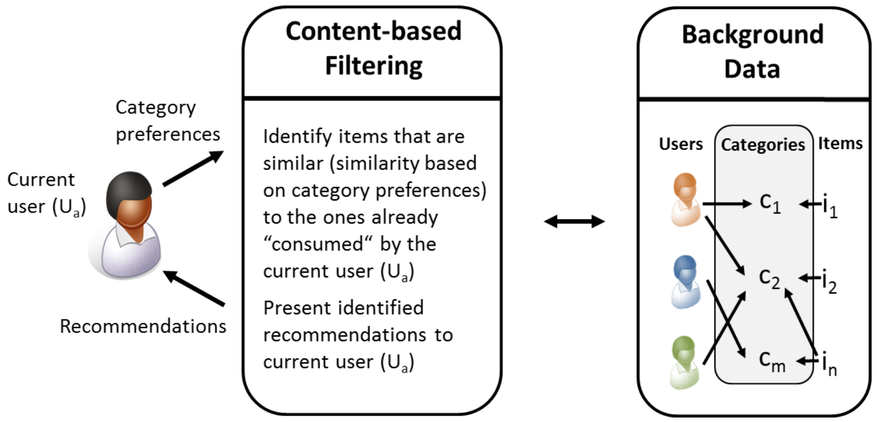
Content-based Filtering
Identify items that are similar (similarity based on category preferences) to the ones already "consumed" by the current user (U_a)
Present identified recommendations to current user (U_a)
Background Data
Users
Categories
Items
c1
c2
cm
i1
i2
in
Category preferences
Current user (U_a)
Recommendations

Fig. 2: Content-based filtering (CBF) dataflow: users are rating items and receive recommendations of items similar to those that have received a good evaluation from the current user ($U_a$).

picted in Table 2. Since user $U_a$ already rated the items $l_4$, $l_6$, and $l_{10}$ (see Table 2), we can infer that $U_a$ is interested in the categories *UML*, *Management*, and *Quality* (see Table 5) where items related to the category *UML* and *Management* have been evaluated two times and items related to *Quality* have been evaluated once.

| category | $U_1$ | $U_2$ | $U_3$ | $U_4$ | $U_a$ |
|---|---|---|---|---|---|
| Java | 3 (yes) | | | 1 (yes) | |
| UML | 3 (yes) | 4 (yes) | 3 (yes) | 3 (yes) | 2 (yes) |
| Management | | 3 (yes) | 3 (yes) | | 2 (yes) |
| Quality | | | | 1 (yes) | 1 (yes) |

Table 5: Degree of interest in different categories, for example, user $U_1$ accessed a learning unit related to the category *Java* three times. If a user accessed an item at least once, it is inferred that the user is interested in this item.

If we are interested in an item recommendation for the user $U_a$ we have to search for those items which are most similar to the items that have already been consumed (evaluated) by the $U_a$. This relies on the simple similarity metric shown in Formula 5 (Dice coefficient which is a variation of the Jaccard coefficient "intensively" taking into account category commonalities – see also Jannach et al. [32]). The major difference to the similarity metrics introduced in the context of collaborative filtering is that in this case similarity is measured using keywords (in contrast to ratings).

$$similarity(U_a, item) = \frac{2 * |categories(U_a) \cap categories(item)|}{|categories(U_a)| + |categories(item)|} \tag{5}$$

| LU | rating ($U_a$) | name | Java | UML | Management | Quality | similarity($U_a, l_i$) |
|---|---|---|---|---|---|---|---|
| $l_1$ | | Data Structures in Java | yes | yes | | | $\frac{2}{5}$ |
| $l_2$ | | Object Relational Mapping | yes | yes | | | $\frac{2}{5}$ |
| $l_3$ | | Software Architectures | | yes | | | $\frac{2}{4}$ |
| $l_4$ | 4.0 | Project Management | | yes | yes | | – |
| $l_5$ | | Agile Processes | | | yes | | $\frac{2}{4}$ |
| $l_6$ | 4.0 | Object Oriented Analysis | | yes | yes | | – |
| $l_7$ | | Object Oriented Design | yes | yes | | | $\frac{2}{5}$ |
| $l_8$ | | UML and the UP | | yes | yes | | $\mathbf{\frac{4}{5}}$ |
| $l_9$ | | Class Diagrams | | yes | | | $\frac{2}{4}$ |
| $l_{10}$ | 3.0 | OO Complexity Metrics | | | | yes | – |
| user $U_a$ | | | | yes | yes | yes | |

Table 6: Example of content-based filtering: *user* $U_a$ has already consumed the items $l_4$, $l_6$, and $l_{10}$ (see Table 2). The item most similar (see Formula 5) to the preferences of $U_a$ is *l8* and is now the best recommendation candidate for the current user.

## 4 Knowledge-based Recommendation

Compared to the approaches of collaborative filtering and content-based filtering, *knowledge-based recommendation* [4, 13, 15, 23, 40] does not primarily rely on item ratings and textual item descriptions but on deep knowledge about the offered items. Such deep knowledge (semantic knowledge [15]) describes an item in more detail and thus allows for a different recommendation approach (see Table 7).

| LU | name | obligatory | duration | semester | complexity | topics | eval |
|---|---|---|---|---|---|---|---|
| $l_1$ | Data Structures in Java | yes | 2 | 2 | 3 | Java,UML | 4.5 |
| $l_2$ | Object Relational Mapping | yes | 3 | 3 | 4 | Java,UML | 4.0 |
| $l_3$ | Software Architectures | no | 3 | 4 | 3 | UML | 3.3 |
| $l_4$ | Project Management | yes | 2 | 4 | 2 | UML,Management | 5.0 |
| $l_5$ | Agile Processes | no | 1 | 3 | 2 | Management | 3.0 |
| $l_6$ | Object Oriented Analysis | yes | 2 | 2 | 3 | UML,Management | 4.7 |
| $l_7$ | Object Oriented Design | yes | 2 | 2 | 3 | Java,UML | 4.0 |
| $l_8$ | UML and the UP | no | 3 | 3 | 2 | UML,Management | 2.0 |
| $l_9$ | Class Diagrams | yes | 4 | 3 | 3 | UML | 3.0 |
| $l_{10}$ | OO Complexity Metrics | no | 3 | 4 | 2 | Quality | 5.0 |

Table 7: Software engineering learning units (LU) described based on *deep knowledge*, for example, *obligatory* vs. *non-obligatory*, *duration* of consumption, recommended *semester*, *complexity* of the learning unit, associated *topics*, and average user rating (*eval*).

Knowledge-based recommendation (see Figure 3) relies on the following background data: (a) a set of rules (constraints) or similarity metrics and (b) a set of items. Depending on the given user requirements, rules (constraints) describe which items have to be recommended. The current user ($U_a$) articulates his/her requirements (preferences) in terms of item property specifications which are internally as well represented in terms of rules (constraints). In our example, constraints are represented solely by user requirements, no further constraint types are included (e.g., constraints that explicitly specify compatibility or incompatibility relationships). An example of such a constraint is *topics* = *Java*. It denotes the fact that the user is primarily interested in Java-related learning units. For a detailed discussion of further constraint types we refer the reader to Felfernig et al. [15]. Constraints are interpreted and the resulting items are presented to the user.[2] A detailed discussion of reasoning mechanisms that are used in knowledge-based recommendation can be found, for example, in Felfernig et al. [15, 16, 22].

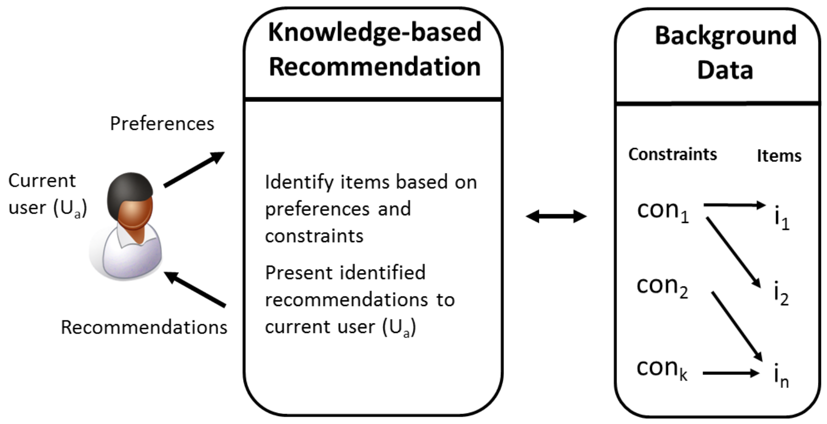
Knowledge-based Recommendation
Background Data
Preferences
Current user ($U_a$)
Recommendations
Identify items based on preferences and constraints
Present identified recommendations to current user ($U_a$)
Constraints
Items
$con_1$
$con_2$
$con_k$
$i_1$
$i_2$
$i_n$

Fig. 3: Knowledge-based recommendation (KBR) dataflow: users are entering their preferences and receive recommendations based on the interpretation of a set of rules (constraints).

In order to determine a recommendation in the context of knowledge-based recommendation scenarios, a *recommendation task* has to be solved.

*Definition (Recommendation Task)*. A recommendation task can be defined by the tuple (R,I) where R represents a set of user requirements and I represents a set of items (in our case: software engineering learning units $l_i \in LU$). The goal is to identify those items in I which fulfill the given user requirements (preferences).

A solution for a recommendation task (also denoted as recommendation) can be defined as follows.

*Definition (Solution for a Recommendation Task)*. A solution for a recommendation task (R,I) is a set S $\subseteq$ I such that $\forall l_i \in S : l_i \in \sigma_{(R)} I$ where $\sigma$ is the selection operator of a conjunctive query [16], R represents a set of selection criteria (represented as constraints), and I represents an item table (see, e.g., Table 7). If we want to

[2] Knowledge-based recommendation approaches based on the determination of similarities between items will be discussed in Section 7.

restrict the set of item properties shown to the user in a result set (recommendation), we have to additionally include projection criteria $\pi$ as follows: $\pi_{(attributes(I))}(\sigma_{(R)}I)$.

In our example, we show how to determine a solution for a given recommendation task based on a conjunctive query where user requirements are used as selection criteria (constraints) on an item table I. If we assume that the user requirements are represented by the set R = $\{r_1 : semester \leq 3, r_2 : topics = Java\}$ and the item table I consists of the elements shown in Table 7 then $\pi_{(LU)}(\sigma_{(semester \leq 3 \wedge topics=Java)}I)$ = $\{l_1, l_2, l_7\}$, i.e., these three items are consistent with the given set of requirements.

*Ranking items*. Up to this point we only know which items can be recommended to a user. One wide-spread approach to rank items is to define a utility scheme which serves as a basis for the application of multi attribute utility theory (MAUT).[3] Alternative items can be evaluated and ranked with respect to a defined set of interest dimensions. In the domain of e-learning units, example interest dimensions of users could be *time effort* (time needed to consume the learning unit) and *quality* (quality of the learning unit). The first step to establish a MAUT scheme is to relate the interest dimensions with properties of the given set of items. A simple example of such a mapping is shown in Table 8. In this example, we assume that obligatory learning units (learning units that have to be consumed within the scope of a study path) trigger more time efforts than non-obligatory ones, a longer duration of a learning unit is correlated with higher time efforts, and low complexity correlates with lower time efforts. In this context, lower time efforts for a learning unit are associated with a higher utility. Furthermore, we assume that the more advanced the semester, the higher is the quality of the learning unit (e.g., in terms of education degree). The better the overall evaluation (eval), the higher the quality of a learning unit (e.g., in terms of the used pedagogical approach).

We are now able to determine the user-specific utility of each individual item. The calculation of *item* utilities for a specific user $U_a$ can be based on Formula 6.

$$utility(U_a, item) = \sum_{d \in Dimensions} contribution(item, d) \times weight(U_a, d) \tag{6}$$

If we assume that the current user $U_a$ assigns a weight of 0.2 to the dimension *time effort* (weight($U_a$,time effort)=0.2) and a weight of 0.8 to the dimension *quality* (weight($U_a$,quality)=0.8) then the user-specific utilities of the individual items ($l_i$) are the ones shown in Table 9.

*Dealing with Inconsistencies*. Due to the logical nature of knowledge-based recommendation problems, we have to deal with scenarios where no solution (recommendation) can be identified for a given set of user requirements, i.e., $\sigma_{(R)}I = \emptyset$. In such situations we are interested in proposals for requirements changes such that a solution (recommendation) can be identified. For example, if a user is interested in learning units with a duration of 4 hours, related to management, and a complexity

[3] A detailed discussion of the application of MAUT in knowledge-based recommendation scenarios can be found in [1, 15, 17].

| item property | time effort [1–10] | quality [1–10] |
|---|---|---|
| obligatory = yes | 4 | - |
| obligatory = no | 7 | - |
| duration = 1 | 10 | - |
| duration = 2 | 5 | - |
| duration = 3 | 1 | - |
| duration = 4 | 1 | - |
| complexity = 2 | 8 | - |
| complexity = 3 | 5 | - |
| complexity = 4 | 2 | - |
| semester = 2 | - | 3 |
| semester = 3 | - | 5 |
| semester = 4 | - | 7 |
| eval = 0–2 | - | 2 |
| eval = >2–3 | - | 5 |
| eval = >3–4 | - | 8 |
| eval = >4 | - | 10 |

Table 8: Contributions of item properties to the dimensions *time effort* and *quality*.

| LU | time effort | quality | utility |
|---|---|---|---|
| $l_1$ | 14 | 13 | 2.8+10.4=13.2 |
| $l_2$ | 7 | 13 | 1.4+10.4=11.8 |
| $l_3$ | 13 | 15 | 2.6+12.0=14.6 |
| $l_4$ | 17 | 17 | 3.4+13.6=**17.0** |
| $l_5$ | 25 | 10 | 5.0+8.0=13.0 |
| $l_6$ | 14 | 13 | 2.8+10.4=13.2 |
| $l_7$ | 14 | 11 | 2.8+8.8=11.6 |
| $l_8$ | 16 | 7 | 3.2+5.6=8.8 |
| $l_9$ | 10 | 10 | 2.0+8.0=10.0 |
| $l_{10}$ | 16 | 17 | 3.2 + 13.6 = 16.8 |

Table 9: Item-specific utility for user $U_a$ (utility($U_a, l_i$)) assuming the personal preferences for *time effort = 0.2* and *quality = 0.8*. In this scenario, item $l_4$ has the highest utility for user $U_a$.

level $> 3$, then no solution can be provided for the given set of requirements R = $\{r_1 : duration = 4, r_2 : topics = management, r_3 : complexity > 3\}$.

User support in such situations can be based on the concepts of conflict detection [33] and model-based diagnosis [12, 14, 49]. A conflict (or conflict set) with regard to an item set I in a given set of requirements R can be defined as follows.

*Definition (Conflict Set).* A conflict set is a set CS $\subseteq$ R such that $\sigma_{(CS)} I = \emptyset$. CS is minimal if there does not exist a conflict set CS' with CS' $\subset$ CS.

In our running example we are able to determine the following minimal conflict sets $CS_i$: $CS_1 : \{r_1, r_2\}, CS_2 : \{r_2, r_3\}$. We will not discuss algorithms that support the determination of minimal conflict sets but refer the reader to the work of Junker [33] who introduces a divide-and-conquer based algorithm with a logarithmic complexity in terms of the needed number of consistency checks.

Based on the identified minimal conflict sets, we are able to determine the corresponding (minimal) diagnoses. A diagnosis for a given set of requirements which is inconsistent with the underlying item table can be defined as follows.

*Definition (Diagnosis).* A diagnosis for a set of requirements R = $\{r_1, r_2, ..., r_n\}$ is a set $\Delta \subseteq R$ such that $\sigma_{(R-\Delta)} I \neq \emptyset$. A diagnosis $\Delta$ is minimal if there does not exist a diagnosis $\Delta'$ with $\Delta' \subset \Delta$.

In other words, a diagnosis (hitting set) is a minimal set of requirements that have to be deleted from R such that a solution can be found for R - $\Delta$. The determination of the complete set of diagnoses for a set of requirements inconsistent with the underlying item table (the corresponding conjunctive query results in $\emptyset$) is based on the construction of hitting set trees [49]. An example of the determination of minimal diagnoses is depicted in Figure 4. There are two possibilities of resolving the conflict set $CS_1$. If we decide to delete the requirement $r_2$, $\sigma_{(\{r_1,r_3\})} I \neq \emptyset$, i.e., a diagnosis has been identified ($\Delta_1 = \{r_2\}$) and – as a consequence – all $CS_i$ have been resolved. Choosing the other alternative and resolving $CS_1$ by deleting $r_1$ does not result in a diagnosis since the conflict $CS_2$ is not resolved. Resolving $CS_2$ by deleting $r_2$ does not result in a minimal diagnosis, since $r_2$ already represents a diagnosis. The second (and last) minimal diagnosis that can be identified in our running example is $\Delta_2 = \{r_1, r_3\}$. For a detailed discussion of the underlying algorithm and analysis we refer the reader to Reiter [49]. Note that a diagnosis provides a hint to which requirements have to be changed. For a discussion of how requirement repairs (change proposals) are calculated, we refer the reader to Felfernig et al. [16].

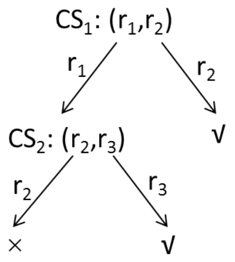

Fig. 4: Determination of the complete set of diagnoses (hitting sets) $\Delta_i$ for the given conflict sets $CS_1 = \{r_1, r_2\}$ and $CS_2 = \{r_2, r_3\}$: $\Delta_1 = \{r_2\}$ and $\Delta_2 = \{r_1, r_3\}$.

## 5 Hybrid Recommendations

After having discussed the three basic recommendation approaches of collaborative filtering, content-based filtering, and knowledge-based recommendation, we will now present some possibilities to combine these basic types.

The motivation for **hybrid recommendation** is the opportunity to achieve a better accuracy [5]. There are different approaches to evaluate the accuracy of recommendation algorithms. These approaches (see also Chapters [**?** **?** ]) can be categorized into *predictive accuracy metrics* such as the mean absolute error (MAE), *classification accuracy metrics* such as precision and recall, and *rank accuracy metrics* such as Kendall's Tau. For a discussion of accuracy metrics we refer the reader also to Gunawardana and Shani [27] and Jannach et al. [32].

We now take a look at example design types of hybrid recommendation approaches [5, 32] which are *weighted*, *mixed*, and *cascade* (see Table 10). These approaches will be explained on the basis of our running example. The basic assumption in the following is that individual recommendation approaches return a list of *five* recommended items where each item has an assigned (recommender-individual) prediction out of $\{1.0, 2.0, 3.0, 4.0, 5.0\}$. For a more detailed discussion of hybridization strategies we refer the reader to Burke [5] and Jannach et al. [32].

| method | description | example formula |
|---|---|---|
| weighted | predictions (s) of individual recommenders are summed up | score(item)=$\Sigma_{rec \in RECS}\ s(item, rec)$ |
| mixed | recommender-individual predictions (s) are combined into one recommendation result | score(item) = zipper-function(item, RECS) |
| cascade | the prediction of one recommender is used as input for the next recommender | score(item) = score(item,$rec_n$) <br> score(item,$rec_i$) $= \begin{cases} s(\text{item},rec_1), & \text{if } i = 1 \\ s(\text{item},rec_i) * \text{score}(\text{item},rec_{i-1}), & \text{otherwise.} \end{cases}$ |

Table 10: Examples of hybrid recommendation approaches (RECS = set of recommenders, s = recommender-individual prediction, score = item score).

*Weighted*. Weighted hybrid recommendation is based on the idea of deriving recommendations by combining the results (predictions) computed by individual recommenders. A corresponding example is depicted in Table 11 where the individual item scores of a collaborative and a content-based recommender are summed up. Item $l8$ receives the highest overall score (9.0) and is ranked highest by the weighted hybrid recommender.[4]

*Mixed*. Mixed hybrid recommendation is based on the idea that predictions of individual recommenders are shown in one integrated result. For example, the re-

[4] If two or more items have the same overall score, a possibility is to force a decision by lot; where needed, this approach can also be applied by other hybrid recommendation approaches.

| items | $l_1$ | $l_2$ | $l_3$ | $l_4$ | $l_5$ | $l_6$ | $l_7$ | $l_8$ | $l_9$ | $l_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| s($l_i$, collaborative filtering) | 1.0 | 3.0 | - | 5.0 | - | 2.0 | - | 4.0 | - | - |
| s($l_i$, content-based filtering) | - | 1.0 | 2.0 | - | - | 3.0 | 4.0 | 5.0 | - | - |
| score($l_i$) | 1.0 | 4.0 | 2.0 | 5.0 | 0.0 | 5.0 | 4.0 | **9.0** | 0.0 | 0.0 |
| ranking($l_i$) | 7 | 4 | 6 | 2 | 8 | 3 | 5 | **1** | 9 | 10 |

Table 11: Example of *weighted* hybrid recommendation: individual predictions are integrated into one score. Item $l8$ receives the best overall score (9.0).

sults of a collaborative filtering and a content-based recommender can be ranked as sketched in Table 12. Item scores can be determined, for example, on the basis of the zipper principle, i.e., the item with highest collaborative filtering prediction value receives the highest overall score (10.0), the item with best content-based filtering prediction value receives the second best overall score, and so forth.

| items | $l_1$ | $l_2$ | $l_3$ | $l_4$ | $l_5$ | $l_6$ | $l_7$ | $l_8$ | $l_9$ | $l_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| s($l_i$, collaborative filtering) | 1.0 | 3.0 | - | 5.0 | - | 2.0 | - | 4.0 | - | - |
| s($l_i$, content-based filtering) | - | 1.0 | 2.0 | - | - | 3.0 | 4.0 | 5.0 | - | - |
| score($l_i$) | 4.0 | 8.0 | 5.0 | **10.0** | 0.0 | 6.0 | 7.0 | 9.0 | 0.0 | 0.0 |
| ranking($l_i$) | 7 | 3 | 6 | **1** | 8 | 5 | 4 | 2 | 9 | 10 |

Table 12: Example of *mixed* hybrid recommendation: individual predictions are integrated into one score conform the zipper principle (best collaborative filtering prediction receives score=10.0, best content-based filtering prediction receives score=9.0 and so forth).

*Cascade*. The basic idea of cascade-based hybridization is that recommenders in a pipe of recommenders exploit the recommendation of the upstream recommender as a basis for deriving their own recommendation. The knowledge-based recommendation approach presented in Section 4 is an example of a cascade-based hybrid recommendation approach. First, items that are consistent with the given requirements are preselected by a conjunctive query $Q$. We can assume, for example, that $s$(item,Q) = 1.0 if the item has been selected and $s$(item,Q) = 0.0 if the item has not been selected. In our case, the set of requirements $\{r_1 : semester \leq 3, r_2 : topics = Java\}$ in the running example leads to the selection of the items $\{l_1, l_2, l_7\}$. Thereafter, these items are ranked conform to their utility for the current user (utility-based ranking $U$). The utility-based ranking ($U$) would determine the item order utility($l_1$) $>$ utility($l_2$) $>$ utility($l_7$) assuming that the current user assigns a weight of 0.8 to the interest dimension *quality* (weight($U_a$,quality) = 0.8) and a weight of 0.2 to the interest dimensions *time effort* (weight($U_a$,time effort) = 0.2). In this example the recommender $Q$ is the first one and the results of $Q$ are forwarded to the utility-based recommender.

Other examples of hybrid recommendation approaches are the following [5]. *Switching* denotes an approach where – depending on the current situation – a specific recommendation approach is chosen. For example, if a user has a low level of product knowledge, then a critiquing-based recommender will be chosen (see Section 7). Vice-versa, if the user is an expert, an interface will be provided where the user is enabled to explicitly state his/her preferences on a detailed level. *Feature combination* denotes an approach where different data sources are exploited by a single recommender. For example, a recommendation algorithm could exploit semantic item knowledge in combination with item ratings (see Table 7). For an in-depth discussion of hybrid recommenders we refer the reader to [5, 32].

## 6 Hints for Practitioners

### *6.1 Usage of Algorithms*

The three basic approaches of collaborative filtering, content-based filtering, and knowledge-based recommendation exploit different sources of recommendation knowledge and have different strengths and weaknesses (see Table 13). Collaborative filtering (CF) as well as content-based filtering (CBF) are easy to set up (only basic item information is needed, e.g., item name and picture) whereas knowledge-based recommendation requires a more detailed specification of item properties (and in many cases also additional constraints). Both, CF and CBF are more adaptive in the sense that new ratings are automatically taken into account in future activations of the recommendation algorithm. In contrast, utility schemes in knowledge-based recommendation (see, for example, Table 9) have to be adapted manually (if no additional learning support is available [21]).

*Serendipity* effects are interpreted as a kind of accident of being confronted with something useful although no related search has been triggered by the user. They can primarily be achieved when using CF approaches. Due to the fact that content-based filtering does not take into account the preferences (ratings) of other users, no such effects can be achieved. Achieving serendipity effects for the users based on KBR is possible in principle, however, restricted to and depending on the creativity of the knowledge engineer (who is able to foresee such effects when defining recommendation rules). The term *ramp-up* problem denotes a situation where there is the need to provide initial rating data before the algorithm is able to determine reasonable recommendations. Ramp-up problems exist with both, CF as well as CBF: in CF users have to rate a set of items before the algorithm is able to determine the nearest neighbors. In CBF, the user has to specify interesting/relevant items before the algorithm is able to determine items that are similar to those that have already been rated by the user.

Finally, *transparency* denotes the degree to which recommendations can be explained to users. Explanations in CF systems solely rely on the interpretation of the relationship to nearest neighbors, for example, *users who purchased item X also purchased item Y*. CBF algorithms explain their recommendations in terms of the similarity of the recommended item to items already purchased by the user: *we recommend Y since you already purchased X which is quite similar to Y*. Finally – due to the fact that they rely on deep knowledge – KBR are able to provide deep explanations which take into account semantic item knowledge. An example of such an explanation are diagnoses which explain the reasons as to why a certain set of requirements does not allow the calculation of a solution. Other types of explanations exist: why a certain item has been included in the recommendation and why a certain question has been asked to the user [15, 18].

Typically, CF and CBF algorithms are used for recommending low-involvement items[5] such as movies, books, and news articles. In contrast, knowledge-based recommender functionalities are used for the recommendation of high-involvement items such as financial services, cars, digital cameras, and apartments. In the latter case, ratings are provided with a low frequency which makes these domains less accessible to CF and CBF approaches. For example, user preferences regarding a car could significantly change within a couple of years without being detected by the recommender system whereas such preference shifts are detected by collaborative and content-based recommendation approaches due to the fact that purchases occur more frequently and – as a consequence – related ratings are available for the recommender system. For an overview of heuristics and rules related to the selection of recommendation approaches we refer the reader to Burke and Ramezani [8].

| algorithm | CF | CBF | KBR |
|---|---|---|---|
| easy setup | yes | yes | no |
| adaptivity | yes | yes | no |
| serendipity effects | yes | no | no |
| ramp-up problem | yes | yes | no |
| transparency | no | no | yes |
| high-involvement items | no | no | yes |

Table 13: Summarization of the strengths and weaknesses of collaborative filtering (CF), content-based filtering (CBF), and knowledge-based recommendation (KBR).

## 6.2 Recommendation Environments

Recommendation is an Artificial Intelligence (AI) technology successfully applied in different commercial contexts [20]. As recommendation algorithms and heuristics are regarded as a major intellectual property of a company, recommender systems are often not developed on the basis of standard solutions but are rather based on proprietary solutions that are tailored to the specific situation of the company. De-

[5] The impact of a wrong decision (selection) is rather low, therefore users invest less evaluation efforts in a purchase situation.

spite this situation, there exist a couple of recommendation environments that can be exploited for the development of different recommender applications.

Strands[6] is a company that provides recommendation technologies covering the whole range of collaborative, content-based, and knowledge based recommendation approaches. MyMediaLite[7] is an open-source library that can be used for the development of collaborative filtering based recommender systems. LensKit[8] [10] is an open source toolkit that supports the development and evaluation of recommender systems – specifically it includes implementations of different collaborative filtering algorithms. A related development is MovieLens[9] which is a non-commercial movie recommendation platform. The MovieLens dataset (user $\times$ item ratings) is publicly available and popular dataset for evaluating new algorithmic developments. Apache Mahout[10] is a machine learning environment that also includes recommendation functionalities such as user-based and item-based collaborative filtering.

Open source constraint libraries such as Choco[11] and Jacop[12] can be exploited for the implementation of knowledge-based recommender applications. WeeVis[13] is a Wiki-based environment for the development of knowledge-based recommender applications – resulting recommender applications can be deployed on different handheld platforms such as iOS, Android, and Windows 8. Finally, Choicla[14] is a group recommendation platform that allows the definition and execution of group recommendation tasks (see Section 7).

## 7 Further Algorithmic Approaches

### 7.1 *Critiquing-based recommendation*

There are two basic approaches to support item identification in the context of knowledge-based recommendation.

First, *search-based* approaches require the explicit specification of search criteria and the recommendation algorithm is in charge of identifying a set of corresponding recommendations [15, 56] (see also Section 4). If no solution can be found for a given set of requirements, the recommendation engine determines diagnoses that indicate potential changes such that a solution (recommendation) can be identified. Second, *navigation-based* approaches support the navigation in the item space

[6] strands.com
[7] www.mymedialite.net
[8] lenskit.grouplens.org
[9] www.movielens.org
[10] mahout.apache.org
[11] www.emn.fr
[12] jacop.osolpro.com
[13] www.weevis.org
[14] choicla.com

where in each iteration a reference item is presented to the user and the user either accepts the (recommended) item or searches for different solutions by specifying *critiques*. Critiques are simple criteria that are used for determining new recommendations that take into account the (changed) preferences of the current user. Examples of such critiques in the context of our running example are *less time efforts* and *higher quality* (see Figure 5). **Critiquing-based recommendation systems** are useful in situations where users are not experts in the item domain and prefer to specify their requirements on the level of critiques [34]. If users are knowledgeable in the item domain, the application of search-based approaches makes more sense. For an in-depth discussion of different variants of critiquing-based recommendation we refer the reader to [7, 9, 26, 39, 44, 50].

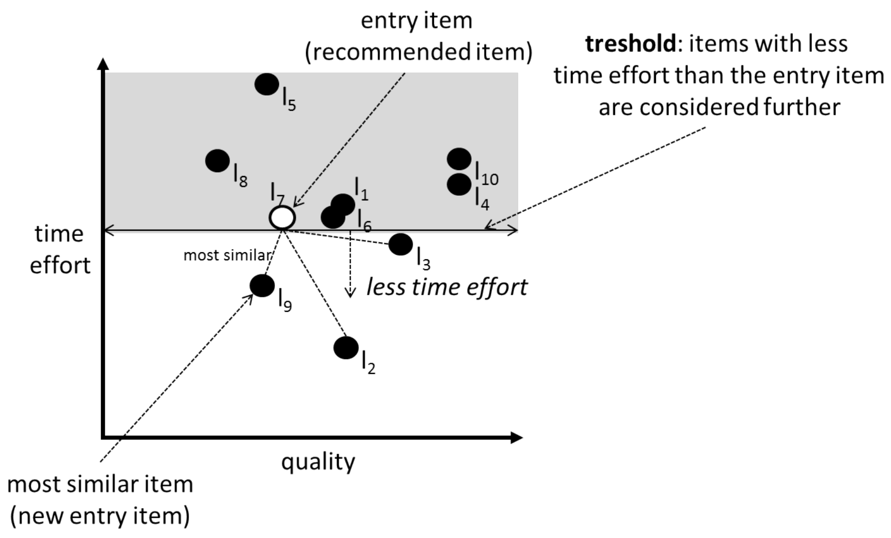

Fig. 5: Example of a critiquing scenario: an entry item ($l_7$) is shown to the user. The user specifies the critique ”less time effort”. The new entry item is $l_9$ since it is consistent with the critique and the item most similar to $l_7$.

## 7.2 *Group recommendation*

Due to the increasing popularity of social platforms and online communities, **group recommendation systems** are becoming an increasingly important technology [28, 42]. Example application domains of group recommendation technologies include tourism [45] (e.g., *which hotels or tourist destinations should be visited by a group?*) and interactive television [41] (*which sequence of television programs will* *be accepted by a group?*). In the majority, group recommendation algorithms are related to simple items such as hotels, tourist destinations, and television programs. The application of group recommendation in the context of our running example is shown in Table 14 (selection of a learning unit for a group).

| items | $l_1$ | $l_2$ | $l_3$ | $l_4$ | $l_5$ | $l_6$ | $l_7$ | $l_8$ | $l_9$ | $l_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| alex | 1.0 | 3.0 | 1.0 | 5.0 | 4.0 | 2.0 | 4.0 | 2.0 | 1.0 | 4.0 |
| dorothy | 5.0 | 1.0 | 2.0 | 1.0 | 4.0 | 3.0 | 4.0 | 2.0 | 2.0 | 3.0 |
| peter | 2.0 | 4.0 | 2.0 | 5.0 | 3.0 | 5.0 | 4.0 | 3.0 | 2.0 | 2.0 |
| ann | 3.0 | 4.0 | 5.0 | 2.0 | 1.0 | 1.0 | 3.0 | 3.0 | 3.0 | 4.0 |
| *least misery* | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | **3.0** | 2.0 | 1.0 | 2.0 |

Table 14: Example of group recommendation: selection of a learning unit for a group. The recommendation ($l_7$) is based on the *least misery* heuristic.

The group recommendation task is it to figure out a recommendation that will be accepted by the whole group. The group decision heuristics applied in the context is *least misery* which returns the lowest voting for alternative $l_i$ as group recommendation. For example, the *least misery* value for alternative $l_7$ is 3.0 which is the highest value of all possible alternatives, i.e., the first recommendation for the group is $l_7$. Other examples of group recommendation heuristics are *most pleasure* (the group recommendation is the item with the highest individual voting) and *majority voting* (the voting for an individual solution is defined by the majority of individual user votings - the group recommendation is the item with the highest majority value). Group recommendation technologies for high-involvement items (see Section 6) are the exception of the rule (see, e.g., [30, 54]). First applications of group recommendation technologies in the software engineering context are reported in Felfernig et al. [24]. An in-depth discussion of different types of group recommendation algorithms can be found in Masthoff [31, 42, 46].

## 8 Conclusions

This chapter provides an introduction to the recommendation approaches of collaborative filtering, content-based filtering, knowledge-based recommendation, and different hybrid variants thereof. While collaborative filtering based approaches exploit ratings of nearest neighbors, content-based filtering exploits categories and/or extracted keywords for determining recommendations. Knowledge-based recommenders should be used, for example, for products where there is a need to encode the recommendation knowledge in terms of constraints. Beside algorithmic approaches, we discussed criteria to be taken into account when deciding about which recommendation technology to use in a certain application context. Furthermore, we provided an overview of environments that can be exploited for recommender application development.

## References

1. L. Ardissono, A. Felfernig, G. Friedrich, A. Goy, D. Jannach, G. Petrone, R. Schäfer, and M. Zanker. A framework for the development of personalized, distributed web-based configuration systems. *AI Magazine*, 24(3):93–108, 2003.
2. R. Bachwani. Preventing and diagnosing software upgrade failures. *PhD Thesis, Rutgers University*, 2012.
3. D. Billsus and M. Pazzani. Learning collaborative information filters. In *15th International Conference on Machine Learning (ICML'98)*, pages 46–54, 1998.
4. R. Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Information Systems*, 69(32):180–200, 2000.
5. R. Burke. Hybrid recommender systems: survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.
6. R. Burke, A. Felfernig, and M. Goeker. Recommender systems: An overview. *AI Magazine*, 32(3):13–18, 2011.
7. R. Burke, K. Hammond, and B. Yound. The findme approach to assisted browsing. *IEEE Expert*, 12(4):32–40, 1997.
8. R. Burke and M. Ramezani. Matching recommendation technologies and domains. *Recommender Systems Handbook*, pages 367–386, 2010.
9. L. Chen and P. Pu. Critiquing-based Recommenders: Survey and Emerging Trends. *User Modeling and User-Adapted Interaction*, 22(1–2):125150, 2012.
10. M.D. Ekstrand, M. Ludwig, J. Kolb, and J. Riedl. LensKit: a modular recommender framework. In *ACM Recommender Systems 2011*, pages 349–350, 2011.
11. M.D. Ekstrand, J.T. Riedl, and J.A. Konstan. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction*, 4(2):1–94, 2011.
12. A. Falkner, A. Felfernig, and A. Haag. Recommendation Technologies for Configurable Products. *AI Magazine*, 32(3):99–108, 2011.
13. A. Felfernig, G. Friedrich, B. Gula, M. Hitz, T. Kruggel, R. Melcher, D. Riepan, S. Strauss, E. Teppan, and O. Vitouch. Persuasive recommendation: Exploring serial position effects in knowledge-based recommender systems. In *2nd International Conference of Persuasive Technology (Persuasive 2007)*, volume 4744 of *LNCS*, pages 283–294. Springer, 2007.
14. A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency-based diagnosis of configuration knowledge bases. *Artificial Intelligence*, 152(2):213–234, February 2004.
15. A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. An Integrated Environment for the Development of Knowledge-based Recommender Applications. *Intl. Journal of Electronic Commerce (IJEC)*, 11(2):11–34, 2006.
16. A. Felfernig, G. Friedrich, M. Schubert, M. Mandl, M. Mairitsch, and E. Teppan. Plausible Repairs for Inconsistent Requirements. In *IJCAI'09*, pages 791–796, Pasadena, CA, 2009.
17. A. Felfernig, B. Gula, G. Leitner, M. Maier, R. Melcher, and E. Teppan. Persuasion in knowledge-based recommendation. In *3rd Intl. Conf. on Persuasive Technology*, LNCS, pages 71–82. Springer, 2008.
18. A. Felfernig, B. Gula, and E. Teppan. Knowledge-based Recommender Technologies for Marketing and Sales. *Special issue of Personalization Techniques for Recommender Systems and Intelligent User Interfaces for the International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 21(2):1–22, 2006.
19. A. Felfernig, K. Isak, K. Szabo, and P. Zachar. The VITA Financial Services Sales Support Environment. In *AAAI/IAAI 2007*, pages 1692–1699, 2007.
20. A. Felfernig, M. Jeran, G. Ninaus, F. Reinfrank, and S. Reiterer. Toward the Next Generation of Recommender Systems. In *Multimedia Services in Intelligent Environments: Recommendation Services*, pages 81–98. Springer, 2013.
21. A. Felfernig, S. Schippel, G. Leitner, F. Reinfrank, K. Isak, M. Mandl, P. Blazek, and G. Ninaus. Automated Repair of Scoring Rules in Constraint-based Recommender Systems. *AI Communications*, 26(2):15–27, 2013.

22. A. Felfernig, M. Schubert, and S. Reiterer. Personalized Diagnosis for Over-Constrained Problems. In *23rd International Conference on Artificial Intelligence*, Peking, China, 2013.
23. A. Felfernig and K. Shchekotykhin. Debugging User Interface Descriptions of Knowledge-based Recommender Applications. In *ACM International Conference on Intelligent User Interfaces (IUI06)*, pages 234–241, Sydney, Australia, 2006.
24. A. Felfernig, C. Zehentner, G. Ninaus, H. Grabner, W. Maaleij, D. Pagano, L. Weninger, and F. Reinfrank. Group Decision Support for Requirements Negotiation. In *Advances in User Modeling*, number 7138 in Lecture Notes in Computer Science, pages 105–116. Springer, 2012.
25. D. Goldberg, D. Nichols, B. Oki, and D. Terry. Using Collaborative Filtering to weave an information Tapestry. *Communications of the ACM*, 35(12):61–70, 1992.
26. P. Grasch, A. Felfernig, and F. Reinfrank. ReComment: Towards Critiquing-based Recommendation with Speech Interaction. In *7th ACM Conference on Recommender Systems*, pages 157–164. ACM, 2013.
27. A. Gunawardana and G. Shani. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research*, 10:2935–2962, 2009.
28. T. Hennig-Thurau, A. Marchand, and P. Marx. Can automated group recommender systems help consumers make better choices? *Journal of Marketing*, 76(5):89–109, 2012.
29. J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In *Conference on Research and Development in Information Retrieval*, pages 230–237, Berkeley, CA, USA, 1999.
30. A. Jameson. More than the sum of its members: Challenges for group recommender systems. In *International Working Conference on Advanced Visual Interfaces*, pages 48–54, 2004.
31. A. Jameson and B. Smyth. Recommendation to groups. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 596–627. 2007.
32. D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems – An Introduction*. Cambridge University Press, 2010.
33. Ulrich Junker. QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In *19th Intl. Conference on Artifical Intelligence*, AAAI'04, pages 167–172. AAAI Press, 2004.
34. B. Knijnenburg, N. Reijmer, and M. Willemsen. Each to his own: how different users call for different interaction methods in recommender systems. In *RecSys 2011*, pages 141–148, Chicago, IL, USA, 2011.
35. J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3):77–87, 1997.
36. Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer*, 42(8):42–49, 2009.
37. S. Lim, D. Quercia, and A. Finkelstein. Stakenet: Using social networks to analyse the stakeholders of large-scale software projects. pages 295–304, Cape Town, South Africa, 2010. ACM/IEEE.
38. G. Linden, B. Smith, and J. York. Amazon.com Recommendations – Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.
39. M. Mandl and A. Felfernig. Improving the Performance of Unit Critiquing. In *20th International Conference on User Modeling, Adaptation, and Personalization (UMAP 2012)*, pages 176–187, Montreal, Canada, 2012.
40. M. Mandl, A. Felfernig, E. Teppan, and M. Schubert. Consumer Decision Making in Knowledge-based Recommendation. *Journal of Intelligent Information Systems (JIIS)*, 37(1):1–22, 2010.
41. J. Masthoff. Group Modeling: Selecting a Sequence of Television Items to Suit a Group of Viewers. *User Modeling and User-Adapted Interaction (UMUAI)*, 14(1):37–85, 2004.
42. J. Masthoff. Group recommender systems: Combining individual models. *Recommender Systems Handbook*, pages 677–702, 2011.

43. F. McCarey, M. Cinneide, and N. Kushmerick. Rascal – A Recommender Agent for Agile Reuse. *Artificial Intelligence Review*, 24(3–4):253–273, 2005.
44. K. McCarthy, J. Reilly, L. McGinty, and B. Smyth. On the dynamic generation of compound critiques in conversational recommender systems. In *Adaptive Hypermedia and Adaptive Web-Based Systems*. Springer, 2004.
45. K. McCarthy, M. Salamo, L. Coyle, L. McGinty, B. Smyth, and P. Nixon. Group recommender systems: a critiquing based approach. In *International Conference on Intelligent User Interface (IUI 2006)*, pages 267–269, Sydney, Australia, 2006.
46. M. O'Connor, D. Cosley, J. Konstan, and J. Riedl. PolyLens: a recommender system for groups of users. In *7th European Conference on Computer Supported Cooperative Work*, pages 199–218, 2001.
47. M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27:313–331, 1997.
48. B. Peischl, M. Zanker, M. Nica, and W. Schmid. Constraint-based Recommendation for Software Project Effort Estimation. *Journal of Emerging Technologies in Web Intelligence*, 2(4):282–290, 2010.
49. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, April 1987.
50. F. Ricci and Q. Nguyen. Acqiring and revising preferences in a critiquing-based mobile recommender systems. *IEEE Intelligent Systems*, 22(3):22–29, 2007.
51. M. Robillard, R. Walker, and T. Zimmermann. Recommendation Systems for Software Engineering. *IEEE Software*, 27(4):80–86, 2010.
52. A. Sahm and W. Maalej. Switch! recommending artifacts needed next based on personal and shared context. In *Software Engineering (Workshops)*, pages 473–484, 2010.
53. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *10th International Conference on World Wide Web*, pages 285–295, 2001.
54. M. Stettinger, G. Ninaus, M. Jeran, F. Reinfrank, and S. Reiterer. WE-DECIDE: A Decision Support Environment for Groups of Users. In *IEA/AIE 2013*, pages 382–391, 2013.
55. G. Takacs, I. Pilaszy, B. Nemeth, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, 2009.
56. J. Tiihonen and A. Felfernig. Towards recommending configurable offerings. *International Journal of Mass Customization*, 3(4):389–406, 2010.
57. M. Tsunoda, T. Kakimoto, N. Ohsugi, A. Monden, and K. Matsumoto. Javawock: A java class recommender system based on collaborative filtering. In *SEKE 2005*, pages 491–497, Taipei, Taiwan, 2005.