Chapter 2
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.

2.1 Introduction

Recommendation systems [7,33] 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*? [39], *which website to visit next*? [49], 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*? [3], *which requirements to implement in the next software release*? [25], *which stakeholders should participate in the upcoming software project*? [38], *which method calls might be useful in the current development context*? [59], *which software components (or APIs) to reuse?* [45], *which software artifacts are needed next*? [40], and *which effort estimation methods should be applied in the current project phase*? [50]. An overview of the application of different types of recommendation technologies in the software engineering context can be found in Robillard et al. [53].

A. Felfernig (✉) • M. Jeran • G. Ninaus • F. Reinfrank • S. Reiterer • M. Stettinger
Institute for Software Technology, Graz University of Technology,
Inffeldgasse 16b/2, 8010 Graz, Austria
e-mail: alexander.felfernig@ist.tugraz.at; mjeran@ist.tugraz.at; gninaus@ist.tugraz.at; florian.reinfrank@ist.tugraz.at; reiterer@ist.tugraz.at; mstettinger@ist.tugraz.at

The major goal of this book chapter is to shed light on the basic properties of the three major recommendation approaches of (1) collaborative filtering [12,26,36], (2) content-based filtering [49], and (3) knowledge-based recommendation [5, 16]. 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 Sect. 2.2 we give an overview of collaborative filtering recommendation approaches. In Sect. 2.3 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 Sect. 2.4). In Sect. 2.5, 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 Sect. 2.6. A short overview of further algorithmic approaches is presented in Sect. 2.7.

2.2 Collaborative Filtering

The item-set in our running examples is *software engineering-related learning material* offered, for example, on an e-learning platform (see Table 2.1). Each learning unit is additionally assigned to a set of categories, for example, the learning unit l_1 is characterized by Java and UML.

Collaborative filtering [12, 36, 56] 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 [39]), family members and friends are replaced by the so-called *nearest neighbors* (NN) who are users with a similar preference pattern or purchasing behavior compared to the current user. Collaborative filtering (see Fig. 2.1) relies on two different types of *background data*: (1) a set of users and (2) 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.

The basic procedure of collaborative filtering can best be explained based on a running example (see Table 2.2) which is taken from the software engineering domain (collaborative recommendation of learning units). Note that in this chapter we focus on the so-called memory-based approaches to collaborative filtering which—in contrast to model-based approaches—operate on uncompressed versions of the user/item matrix [4]. The two basic approaches to collaborative filtering are *user-based collaborative filtering* [36] and *item-based collaborative filtering* [54]. 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.

Table 2.1 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

Learning unit	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

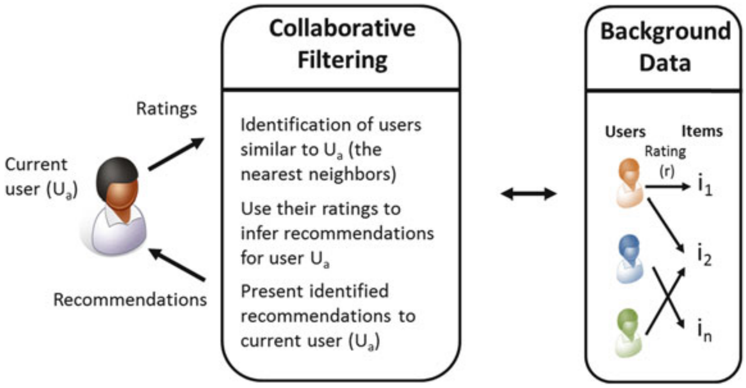
Collaborative Filtering
Identification of users similar to U_a (the nearest neighbors)
Use their ratings to infer recommendations for user U_a
Present identified recommendations to current user (U_a)
Background Data
Users
Items
Rating (r)
i1
i2
in
Current user (U_a)
Ratings
Recommendations

Fig. 2.1 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)

User-Based Collaborative Filtering. User-based collaborative filtering identifies the k-nearest neighbors of the active user—see Eq. (2.1)[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 2.2, user U_2 is the nearest neighbor ($k = 1$) of user U_a, based on Eq. (2.1), and his/her rating of learning unit l_3 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 [33]; see Eq. (2.1), where LU_c is the set of items that have been rated by both users, r_{α,l_i} is the rating of user α for item l_i, and

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

Table 2.2 Example collaborative filtering data structure (rating matrix): learning units (LU) versus related user ratings (we assume a rating scale of 1–5)

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 2.3 Similarity between user U_a and the users $U_j \neq U_a$ determined based on Eq. (2.1). If the number of commonly rated items is below 2, no similarity between the two users is calculated

	U_1	U_2	U_3	U_4
U_a	–	0.97	0.70	–

$\overline{r_\alpha}$ is the average rating of user α. Similarity values resulting from the application of Eq. (2.1) can take values on a scale of $[-1, \ldots, +1]$.

$$\text{similarity}(U_a, U_x) = \frac{\sum_{l_i \in LU_c} (r_{a,l_i} - \overline{r_a}) \times (r_{x,l_i} - \overline{r_x})}{\sqrt{\sum_{l_i \in LU_c} (r_{a,l_i} - \overline{r_a})^2} \times \sqrt{\sum_{l_i \in LU_c} (r_{x,l_i} - \overline{r_x})^2}} \tag{2.1}$$

The similarity values for U_a calculated based on Eq. (2.1) are shown in Table 2.3. For the purposes of our example we assume the existence of at least two items per user pair (U_i, U_j), for $i \neq j$, in order to be able to determine a similarity. This criterion holds for users U_2 and U_3.

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* [30], i.e., the higher the number of commonly rated items, the higher is the significance of

Table 2.4 User-based collaborative filtering-based recommendations (predictions) for items that have not been rated by user U_a up to now

	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(U_a, l_i)	–	–	**3.045**	–	–	–	–	**2.045**	–	–

the corresponding correlation. For further information regarding the handling of sparsity, we refer the reader to [30, 33].

The information about the set of users with a similar rating behavior compared to the current user (*NN*, the set of nearest neighbors) 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 Eq. (2.2).

$$\text{prediction}(U_a, item) = \overline{r_a} + \frac{\sum_{U_j \in NN} \text{similarity}(U_a, U_j) \times (r_{j,item} - \overline{r_j})}{\sum_{U_j \in NN} \text{similarity}(U_a, U_j)} \tag{2.2}$$

Based on the rating of the nearest neighbor of U_a, we are able to determine a prediction for user U_a (see Table 2.4). The nearest neighbor of U_a is user U_2 (see Table 2.3). The learning units rated by U_2 but not rated by U_a are l_3 and l_8. Due to the determined predictions—Eq. (2.2)—item l_3 would be ranked higher than item l_8 in a recommendation list.

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 l_4 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 l_4 and has not been rated by U_a is item l_3 (similarity$(l_3, l_4) = 0.35$). In this case, the nearest neighbor of item l_3 is l_4; this calculation is based on Eq. (2.3).

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.

$$\text{similarity}(l_a, l_b) = \frac{\sum_{u \in U} (r_{u,l_a} - \overline{r_{l_a}}) \times (r_{u,l_b} - \overline{r_{l_b}})}{\sqrt{\sum_{u \in U} (r_{u,l_a} - \overline{r_{l_a}})^2} \times \sqrt{\sum_{u \in U} (r_{u,l_b} - \overline{r_{l_b}})^2}} \tag{2.3}$$

The information about the set of items with a similar rating pattern compared to the *item* under consideration is the basis for predicting the rating of user U_a for the *item*; see Eq. (2.4). 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.

$$\text{prediction}(U_a, item) = \frac{\sum_{it \in NN} \text{similarity}(item, it) \times r_{a,it}}{\sum_{it \in NN} \text{similarity}(item, it)} \quad (2.4)$$

For a discussion of advanced collaborative recommendation approaches, we refer the reader to Koren et al. [37] and Sarwar et al. [54].

2.3 Content-Based Filtering

Content-based filtering [49] 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 [49] (news items with a similar content compared to the set of already consumed news).

Content-based filtering (see Fig. 2.2) relies on two different types of background data: (1) a set of users and (2) 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. [33]. Within the scope of this chapter we focus on content-based recommendation which exploits item categories (see Table 2.1).

Content-based filtering will now be explained based on a running example which relies on the information depicted in Tables 2.1, 2.5, and 2.6. Table 2.1 provides an overview of the relevant items and the assignments of items to categories. Table 2.5 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 depicted in Table 2.2. Since user U_a already rated the items l_4, l_6, and l_{10} (see Table 2.2), we can infer that U_a is interested in the categories UML, Management, and Quality (see Table 2.5) where items related to the category UML and Management have been evaluated two times and items related to Quality have been evaluated once.

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

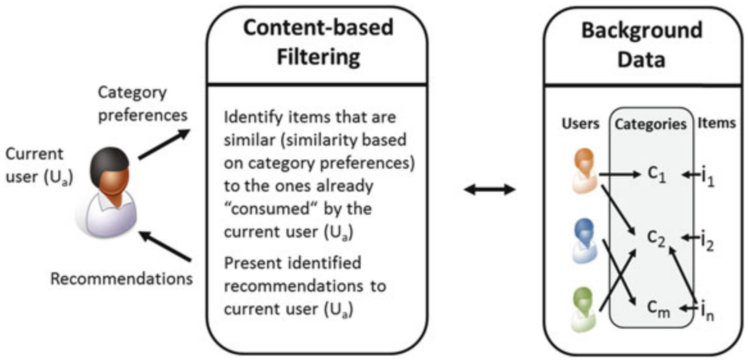
Category preferences
Current user (U_a)
Recommendations
Content-based Filtering
Identify items that are similar (similarity based on category preferences) to the ones already "consumed" by the current user (U_a)
Present identified recommendations to current user (U_a)
Background Data
Users
Categories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

Fig. 2.2 Content-based filtering (CBF) dataflow. Users rate items and receive recommendations of items similar to those that have received a good evaluation from the current user U_a

Table 2.5 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

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)

(evaluated) by the U_a. This relies on the simple similarity metric shown in Eq. (2.5) (the Dice coefficient, which is a variation of the Jaccard coefficient that "intensively" takes into account category commonalities—see also Jannach et al. [33]). The major difference from the similarity metrics introduced in the context of collaborative filtering is that in this case similarity is measured using keywords (in contrast to ratings).

$$\text{similarity}(U_a, item) = \frac{2 \times \text{categories}(U_a) \ \cap \ \text{categories}(item)}{\text{categories}(U_a) + \text{categories}(item)} \tag{2.5}$$

2.4 Knowledge-Based Recommendation

Compared to the approaches of collaborative filtering and content-based filtering, *knowledge-based recommendation* [5, 14, 16, 23, 42] does not primarily rely on item ratings and textual item descriptions but on deep knowledge about the offered items. Such deep knowledge (semantic knowledge [16]) describes an item in more detail and thus allows for a different recommendation approach (see Table 2.7).

Table 2.6 Example of content-based filtering. User U_a has already consumed the items l_4, l_6, and l_{10}; see Table 2.2. The item most similar—see Eq. (2.5)—to the preferences of U_a is l_8 and is now the best recommendation candidate for the current user

LU	Rating of U_a	Name	Java	UML	Management	Quality	similarity(U_a, l_i)
l_1		Data Structures in Java	yes	yes			2/5
l_2		Object Relational Mapping	yes	yes			2/5
l_3		Software Architectures		yes			2/4
l_4	4.0	Project Management		yes	yes		–
l_5		Agile Processes			yes		2/4
l_6	4.0	Object Oriented Analysis		yes	yes		–
l_7		Object Oriented Design	yes	yes			2/5
l_8		UML and the UP		yes	yes		**4/5**
l_9		Class Diagrams		yes			2/4
l_{10}	3.0	OO Complexity Metrics				yes	–
U_a				yes	yes	yes	

Table 2.7 Software engineering learning units (LU) described based on *deep knowledge*: obligatory vs. nonobligatory (Oblig.), duration of consumption (Dur.), recommended semester (Sem.), complexity of the learning unit (Compl.), associated topics (Topics), and average user rating (Eval.)

LU	Name	Oblig.	Dur.	Sem.	Compl.	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

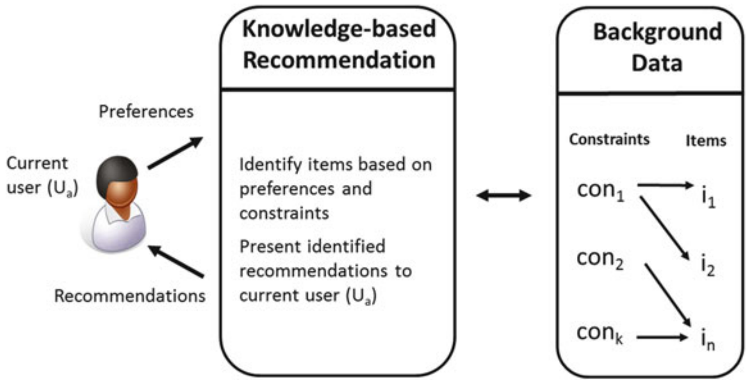
Knowledge-based Recommendation
Identify items based on preferences and constraints
Present identified recommendations to current user (U_a)
Preferences
Current user (U_a)
Recommendations
Background Data
Constraints
Items
con_1
con_2
con_k
i_1
i_2
i_n

Fig. 2.3 Knowledge-based recommendation (KBR) dataflow: users are entering their preferences and receive recommendations based on the interpretation of a set of rules (constraints)

Knowledge-based recommendation (see Fig. 2.3) 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. [16]. 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. [16, 17, 22].

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

Definition 2.1. A *recommendation task* is a 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.

[2]Knowledge-based recommendation approaches based on the determination of similarities between items will be discussed in Sect. 2.7.

Definition 2.2. 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 σ is the selection operator of a conjunctive query [17], R represents a set of selection criteria (represented as constraints), and I represents an item table (see, for example, Table 2.7). If we want to restrict the set of item properties shown to the user in a result set (recommendation), we have to additionally include projection criteria π 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 $R = \{r_1 : semester \leq 3, r_2 : topics = \text{Java}\}$ and the item table I consists of the elements shown in Table 2.7, then $\pi_{(LU)}(\sigma_{(semester \leq 3 \, \wedge \, topics = \text{Java})} I) = \{l_1, l_2, l_7\}$, 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 widespread 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 to properties of the given set of items. A simple example of such a mapping is shown in Table 2.8. 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 nonobligatory 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 Eq. (2.6).

$$\text{utility}(U_a, item) = \sum_{d \in Dimensions} \text{contribution}(item, d) \times \text{weight}(U_a, d) \qquad (2.6)$$

If we assume that the current user U_a assigns a weight of 0.2 to the dimension *time effort* ($\text{weight}(U_a, time\ effort) = 0.2$) and a weight of 0.8 to the dimension *quality* ($\text{weight}(U_a, quality) = 0.8$), then the user-specific utilities of the individual items (l_i) are the ones shown in Table 2.9.

[3]A detailed discussion of the application of MAUT in knowledge-based recommendation scenarios can be found in Ardissono et al. [1] and Felfernig et al. [16, 18].

Table 2.8 Contributions of item properties to the dimensions *time effort* and *quality*

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 2.9 Item-specific utility for user U_a (i.e., 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

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	$3.4 + 13.6 = \mathbf{17.0}$
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	$3.2 + 13.6 = 16.8$

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

User support in such situations can be based on the concepts of conflict detection [34] and model-based diagnosis [13, 15, 51]. A conflict (or conflict set)

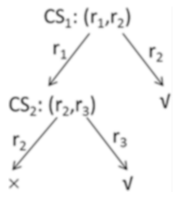

Fig. 2.4 Determination of the complete set of diagnoses (hitting sets) Δ_i for the given conflict sets $CS_1 = \{r_1, r_2\}$ and $CS_2 = \{r_2, r_3\}$: $\Delta_1 = \{r_2\}$ and $\Delta_2 = \{r_1, r_3\}$

with regard to an item set I in a given set of requirements R can be defined as follows.

Definition 2.3. 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 [34] 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 2.4. A *diagnosis* for a set of requirements $R = \{r_1, r_2, \ldots, r_n\}$ is a set $\Delta \subseteq R$ such that $\sigma_{(R-\Delta)} I \neq \emptyset$. A diagnosis Δ is *minimal* if there does not exist a diagnosis Δ' with $\Delta' \subset \Delta$.

In other words, a diagnosis (also called a *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 [51]. An example of the determination of minimal diagnoses is depicted in Fig. 2.4. 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 [51]. 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. [17].

Table 2.10 Examples of hybrid recommendation approaches (*RECS* = set of recommenders, *s* = recommender-individual prediction, score = item score)

Method	Description	Example formula
weighted	predictions of individual recommenders are summed up	$\text{score}(item) = \Sigma_{rec \in RECS}\, s(item, rec)$
mixed	recommender-individual predictions are combined into one recommendation result	$\text{score}(item) = \text{zipper-function}(item, RECS)$
cascade	the predictions of one recommender are used as input for the next recommender	$\text{score}(item) = \text{score}(item, rec_n)$ $\text{score}(item, rec_i) = \begin{cases} s(item, rec_1)\,, & \text{if } i = 1 \\ s(item, rec_i) \times \\ \quad \text{score}(item, rec_{i-1})\,, & \text{otherwise.} \end{cases}$

2.5 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 recommendations is the opportunity to achieve a better accuracy [6]. There are different approaches to evaluate the accuracy of recommendation algorithms. These approaches (see also Avazpour et al. [2] and Tosun Mısırlı et al. [58] in Chaps. 10 and 13, respectively) 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 [28] and Jannach et al. [33].

We now take a look at example design types of hybrid recommendation approaches [6, 33] which are *weighted*, *mixed*, and *cascade* (see Table 2.10). 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 [6] and Jannach et al. [33].

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 2.11 where the

Table 2.11 Example of *weighted* hybrid recommendation: individual predictions are integrated into one score. Item l_8 receives the best overall score (9.0)

Items	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}
$s(l_i, \textit{collaborative filtering})$	1.0	3.0	–	5.0	–	2.0	–	4.0	–	–
$s(l_i, \textit{content-based filtering})$	–	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 2.12 Example of *mixed* hybrid recommendation. Individual predictions are integrated into one score conform the zipper principle (best collaborative filtering prediction receives score = 10, best content-based filtering prediction receives score = 9 and so forth)

Items	l_1	l_2	l_3	l_4	l_5	l_6	l_7	l_8	l_9	l_{10}
$s(l_i, \textit{collaborative filtering})$	1.0	3.0	–	5.0	–	2.0	–	4.0	–	–
$s(l_i, \textit{content-based filtering})$	–	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

individual item scores of a collaborative and a content-based recommender are summed up. Item l_8 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 results of a collaborative filtering and a content-based recommender can be ranked as sketched in Table 2.12. 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.

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 Sect. 2.4 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 $R = \{r_1 : semester \leq 3, r_2 : topics = Java\}$ in the running example leads to the selection of the items $\{l_1, l_2, l_7\}$. Thereafter, these items are ranked conform to

[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.

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 include the following [6]. *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 Sect. 2.7). 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 2.7). For an in-depth discussion of hybrid recommenders, we refer the reader to Burke [6] and Jannach et al. [33].

2.6 Hints for Practitioners

In this section we provide several hints for practitioners who are interested in developing recommendation systems.

2.6.1 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 2.13). Collaborative filtering (CF) and 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 2.9) 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

Table 2.13 Summary of the strengths and weaknesses of collaborative filtering (CF), content-based filtering (CBF), and knowledge-based recommendation (KBR)

Property	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

the creativity of the knowledge engineer (who is able to foresee such effects when defining recommendation rules). The *ramp-up problem* (also called the cold start 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 and 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 is able to provide deep explanations which take into account semantic item knowledge. An example of such an explanation is diagnoses that 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 [16, 24].

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

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

recommender system. For an overview of heuristics and rules related to the selection of recommendation approaches, we refer the reader to Burke and Ramezani [9].

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

Strands is a company that provides recommendation technologies covering the whole range of collaborative, content-based, and knowledge-based recommendation approaches. MyMediaLite is an open-source library that can be used for the development of collaborative filtering-based recommender systems. LensKit [11] 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 which is a noncommercial movie recommendation platform. The MovieLens dataset (user $\times$ item ratings) is publicly available and popular dataset for evaluating new algorithmic developments. Apache Mahout 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 and Jacop can be exploited for the implementation of knowledge-based recommender applications. WeeVis 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 is a group recommendation platform that allows the definition and execution of group recommendation tasks (see Sect. 2.7).

2.7 Further Algorithmic Approaches

We examine two further algorithmic approaches here: general critiquing-based recommendations and group recommendations.

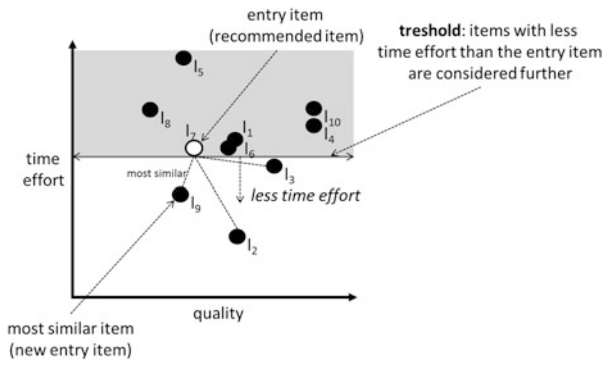

Fig. 2.5 Example of a critiquing scenario. The entry item l_7 is shown to the user. The user specifies the critique "less time effort." The new entry item is l_9 since it is consistent with the critique and the item most similar to l_7

2.7.1 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 [16,57] (see also Sect. 2.4). 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 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 Fig. 2.5). 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 [35]. 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 [8, 10, 27, 41, 46, 52].

2.7.2 Group Recommendation

Due to the increasing popularity of social platforms and online communities, group recommendation systems are becoming an increasingly important technology

Table 2.14 Example of group recommendation: selection of a learning unit for a group. The recommendation (l_7) is based on the *least misery* heuristic

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

[29, 44]. Example application domains of group recommendation technologies include tourism [47] (e.g., *which hotels or tourist destinations should be visited by a group?*) and interactive television [43] (*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 2.14 (selection of a learning unit for a group).

The group recommendation task is 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 most individual votes) and *majority voting* (the voting for an individual solution is defined by the majority of individual user votes: the group recommendation is the item with the highest majority value). Group recommendation technologies for high-involvement items (see Sect. 2.6) are the exception of the rule [e.g., 31, 55]. First applications of group recommendation technologies in the software engineering context are reported in Felfernig et al. [25]. An in-depth discussion of different types of group recommendation algorithms can be found in O'Connor et al. [48], Jameson and Smyth [32], and Masthoff [44].

2.8 Conclusion

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. Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Petrone, G., Schäfer, R., Zanker, M.: A framework for the development of personalized, distributed web-based configuration systems. AI Mag. **24**(3), 93–108 (2003)
2. Avazpour, I., Pitakrat, T., Grunske, L., Grundy, J.: Dimensions and metrics for evaluating recommendation systems. In: Robillard, M., Maalej, W., Walker, R.J., Zimmermann, T. (eds.) Recommendation Systems in Software Engineering, Chap. 10. Springer, New York (2014)
3. Bachwani, R.: Preventing and diagnosing software upgrade failures. Ph.D. thesis, Rutgers University (2012)
4. Billsus, D., Pazzani, M.: Learning collaborative information filters. In: Proceedings of the International Conference on Machine Learning, pp. 46–54 (1998)
5. Burke, R.: Knowledge-based recommender systems. Encyclopedia Libr. Inform. Sci. **69**(32), 180–200 (2000)
6. Burke, R.: Hybrid recommender systems: Survey and experiments. User Model. User-Adapt. Interact. **12**(4), 331–370 (2002). DOI 10.1023/A:1021240730564
7. Burke, R., Felfernig, A., Goeker, M.: Recommender systems: An overview. AI Mag. **32**(3), 13–18 (2011)
8. Burke, R., Hammond, K., Yound, B.: The FindMe approach to assisted browsing. IEEE Expert **12**(4), 32–40 (1997). DOI 10.1109/64.608186
9. Burke, R., Ramezani, M.: Matching recommendation technologies and domains. In: Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds.) Recommender Systems Handbook, pp. 367–386. Springer, New York (2011). DOI 10.1007/978-0-387-85820-3_11
10. Chen, L., Pu, P.: Critiquing-based recommenders: Survey and emerging trends. User Model. User-Adapt. Interact. **22**(1–2), 125–150 (2012). DOI 10.1007/s11257-011-9108-6
11. Ekstrand, M.D., Ludwig, M., Kolb, J., Riedl, J.: LensKit: A modular recommender framework. In: Proceedings of the ACM Conference on Recommender Systems, pp. 349–350 (2011a). DOI 10.1145/2043932.2044001
12. Ekstrand, M.D., Riedl, J.T., Konstan, J.A.: Collaborative filtering recommender systems. Found. Trends Hum. Comput. Interact. **4**(2), 81–173 (2011b). DOI 10.1561/1100000009
13. Falkner, A., Felfernig, A., Haag, A.: Recommendation technologies for configurable products. AI Mag. **32**(3), 99–108 (2011)
14. Felfernig, A., Friedrich, G., Gula, B., Hitz, M., Kruggel, T., Melcher, R., Riepan, D., Strauss, S., Teppan, E., Vitouch, O.: Persuasive recommendation: Serial position effects in knowledge-based recommender systems. In: Proceedings of the International Conference of Persuasive Technology, *Lecture Notes in Computer Science*, vol. 4744, pp. 283–294 (2007a). DOI 10.1007/978-3-540-77006-0_34
15. Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M.: Consistency-based diagnosis of configuration knowledge bases. Artif. Intell. **152**(2), 213–234 (2004). DOI 10.1016/S0004-3702(03)00117-6
16. Felfernig, A., Friedrich, G., Jannach, D., Zanker, M.: An integrated environment for the development of knowledge-based recommender applications. Int. J. Electron. Commerce **11**(2), 11–34 (2006a). DOI 10.2753/JEC1086-4415110201
17. Felfernig, A., Friedrich, G., Schubert, M., Mandl, M., Mairitsch, M., Teppan, E.: Plausible repairs for inconsistent requirements. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 791–796 (2009)

18. Felfernig, A., Gula, B., Leitner, G., Maier, M., Melcher, R., Teppan, E.: Persuasion in knowledge-based recommendation. In: Proceedings of the International Conference on Persuasive Technology, *Lecture Notes in Computer Science*, vol. 5033, pp. 71–82 (2008). DOI 10.1007/978-3-540-68504-3_7
19. Felfernig, A., Isak, K., Szabo, K., Zachar, P.: The VITA financial services sales support environment. In: Proceedings of the Innovative Applications of Artificial Intelligence Conference, pp. 1692–1699 (2007b)
20. Felfernig, A., Jeran, M., Ninaus, G., Reinfrank, F., Reiterer, S.: Toward the next generation of recommender systems: Applications and research challenges. In: Multimedia Services in Intelligent Environments: Advances in Recommender Systems, *Smart Innovation, Systems and Technologies*, vol. 24, pp. 81–98. Springer, New York (2013a). DOI 10.1007/978-3-319-00372-6_5
21. Felfernig, A., Ninaus, G., Grabner, H., Reinfrank, F., Weninger, L., Pagano, D., Maalej, W.: An overview of recommender systems in requirements engineering. In: Managing Requirements Knowledge, Chap. 14, pp. 315–332. Springer, New York (2013b). DOI 10.1007/978-3-642-34419-0_14
22. Felfernig, A., Schubert, M., Reiterer, S.: Personalized diagnosis for over-constrained problems. In: Proceedings of the International Joint Conference on Artificial Intelligence, pp. 1990–1996 (2013c)
23. Felfernig, A., Shchekotykhin, K.: Debugging user interface descriptions of knowledge-based recommender applications. In: Proceedings of the International Conference on Intelligent User Interfaces, pp. 234–241 (2006). DOI 10.1145/1111449.1111499
24. Felfernig, A., Teppan, E., Gula, B.: Knowledge-based recommender technologies for marketing and sales. Int. J. Pattern Recogn. Artif. Intell. **21**(2), 333–354 (2006b). DOI 10.1142/S0218001407005417
25. Felfernig, A., Zehentner, C., Ninaus, G., Grabner, H., Maaleij, W., Pagano, D., Weninger, L., Reinfrank, F.: Group decision support for requirements negotiation. In: Advances in User Modeling, no. 7138 in Lecture Notes in Computer Science, pp. 105–116 (2012). DOI 10.1007/978-3-642-28509-7_11
26. Goldberg, D., Nichols, D., Oki, B., Terry, D.: Using collaborative filtering to weave an information tapestry. Comm. ACM **35**(12), 61–70 (1992). DOI 10.1145/138859.138867
27. Grasch, P., Felfernig, A., Reinfrank, F.: ReComment: Towards critiquing-based recommendation with speech interaction. In: Proceedings of the ACM Conference on Recommender Systems pp. 157–164 (2013)
28. Gunawardana, A., Shani, G.: A survey of accuracy evaluation metrics of recommendation tasks. J. Mach. Learn. Res. **10**, 2935–2962 (2009)
29. Hennig-Thurau, T., Marchand, A., Marx, P.: Can automated group recommender systems help consumers make better choices? J. Market. **76**(5), 89–109 (2012)
30. Herlocker, J., Konstan, J., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: Proceedings of the ACM SIGIR International Conference on Research and Development in Information Retrieval, pp. 230–237 (1999). DOI >10.1145/312624.312682
31. Jameson, A.: More than the sum of its members: Challenges for group recommender systems. In: Proceedings of the Working Conference on Advanced Visual Interfaces, pp. 48–54 (2004). DOI 10.1145/989863.989869
32. Jameson, A., Smyth, B.: Recommendation to groups. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web: Methods and Strategies of Web Personalization, *Lecture Notes in Computer Science*, vol. 4321, Chap. 20, pp. 596–627. Springer, New York (2007). DOI 10.1007/978-3-540-72079-9_20
33. Jannach, D., Zanker, M., Felfernig, A., Friedrich, G.: Recommender Systems: An Introduction. Cambridge University Press, Cambridge (2010)
34. Junker, U.: QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. In: Proceedings of the National Conference on Artifical Intelligence, pp. 167–172 (2004)

35. Knijnenburg, B., Reijmer, N., Willemsen, M.: Each to his own: How different users call for different interaction methods in recommender systems. In: Proceedings of the ACM Conference on Recommender Systems, pp. 141–148 (2011). DOI 10.1145/2043932.2043960
36. Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., Riedl, J.: GroupLens: Applying collaborative filtering to Usenet news. Comm. ACM **40**(3), 77–87 (1997). DOI 10.1145/245108.245126
37. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009). DOI 10.1109/MC.2009.263
38. Lim, S., Quercia, D., Finkelstein, A.: StakeNet: Using social networks to analyse the stakeholders of large-scale software projects. In: Proceedings of the ACM/IEEE International Conference on Software Engineering, pp. 295–304 (2010). DOI 10.1145/1806799.1806844
39. Linden, G., Smith, B., York, J.: Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Comput. **7**(1), 76–80 (2003). DOI 10.1109/MIC.2003.1167344
40. Maalej, W., Sahm, A.: Assisting engineers in switching artifacts by using task semantic and interaction history. In: Proceedings of the International Workshop on Recommendation Systems for Software Engineering, pp. 59–63 (2010). DOI 10.1145/1808920.1808935
41. Mandl, M., Felfernig, A.: Improving the performance of unit critiquing. In: Proceedings of the International Conference on User Modeling, Adaptation, and Personalization, pp. 176–187 (2012). DOI 10.1007/978-3-642-31454-4_15
42. Mandl, M., Felfernig, A., Teppan, E., Schubert, M.: Consumer decision making in knowledge-based recommendation. J. Intell. Inform. Syst. **37**(1), 1–22 (2010). DOI 10.1007/s10844-010-0134-3
43. Masthoff, J.: Group modeling: Selecting a sequence of television items to suit a group of viewers. User Model. User-Adapt. Interact. **14**(1), 37–85 (2004). DOI 10.1023/B:USER.0000010138.79319.fd
44. Masthoff, J.: Group recommender systems: Combining individual models. In: Ricci, F., Rokach, L., Shapira, B., Kantor, P. (eds.) Recommender Systems Handbook, Chap. 21, pp. 677–702. Springer, New York (2011). DOI 10.1007/978-0-387-85820-3_21
45. McCarey, F., Ó Cinnéide, M., Kushmerick, N.: RASCAL: A recommender agent for agile reuse. Artif. Intell. Rev. **24**(3–4), 253–276 (2005). DOI 10.1007/s10462-005-9012-8
46. McCarthy, K., Reilly, J., McGinty, L., Smyth, B.: On the dynamic generation of compound critiques in conversational recommender systems. In: Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, *Lecture Notes in Computer Science*, vol. 3137, pp. 176–184 (2004)
47. McCarthy, K., Salamo, M., Coyle, L., McGinty, L., Smyth, B., Nixon, P.: Group recommender systems: A critiquing based approach. In: Proceedings of the International Conference on Intelligent User Interfaces, pp. 267–269 (2006). DOI 10.1145/1111449.1111506
48. O'Connor, M., Cosley, D., Konstan, J., Riedl, J.: PolyLens: A recommender system for groups of users. In: Proceedings of the European Conference on Computer Supported Cooperative Work, pp. 199–218 (2001). DOI 10.1007/0-306-48019-0_11
49. Pazzani, M., Billsus, D.: Learning and revising user profiles: The identification of interesting web sites. Mach. Learn. **27**(3), 313–331 (1997). DOI 10.1023/A:1007369909943
50. Peischl, B., Zanker, M., Nica, M., Schmid, W.: Constraint-based recommendation for software project effort estimation. J. Emerg. Tech. Web Intell. **2**(4), 282–290 (2010). DOI 10.4304/jetwi.2.4.282-290
51. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987). DOI 10.1016/0004-3702(87)90062-2
52. Ricci, F., Nguyen, Q.: Acqiring and revising preferences in a critiquing-based mobile recommender systems. IEEE Intell. Syst. **22**(3), 22–29 (2007). DOI 10.1109/MIS.2007.43
53. Robillard, M.P., Walker, R.J., Zimmermann, T.: Recommendation systems for software engineering. IEEE Software **27**(4), 80–86 (2010). DOI 10.1109/MS.2009.161
54. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the International Conference on the World Wide Web, pp. 285–295 (2001). DOI 10.1145/371920.372071

55. Stettinger, M., Ninaus, G., Jeran, M., Reinfrank, F., Reiterer, S.: WE-DECIDE: A decision support environment for groups of users. In: Proceedings of the International Conference on Industrial, Engineering, and Other Applications of Applied Intelligent Systems, pp. 382–391 (2013). DOI 10.1007/978-3-642-38577-3_39
56. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Scalable collaborative filtering approaches for large recommender systems. J. Mach. Learn. Res. **10**, 623–656 (2009)
57. Tiihonen, J., Felfernig, A.: Towards recommending configurable offerings. Int. J. Mass Customization **3**(4), 389–406 (2010). DOI 10.1504/IJMASSC.2010.037652
58. Tosun Mısırlı, A., Bener, A., Çağlayan, B., Çalıklı, G., Turhan, B.: Field studies: A methodology for construction and evaluation of recommendation systems in software engineering. In: Robillard, M., Maalej, W., Walker, R.J., Zimmermann, T. (eds.) Recommendation Systems in Software Engineering, Chap. 13. Springer, New York (2014)
59. Tsunoda, M., Kakimoto, T., Ohsugi, N., Monden, A., Matsumoto, K.: Javawock: A Java class recommender system based on collaborative filtering. In: Proceedings of the International Conference on Software Engineering and Knowledge Engineering, pp. 491–497 (2005)