MODULE II

Knowledge-based recommendation

Most commercial recommender systems in practice are based on collaborative filtering (CF) techniques. CF systems rely solely on the user ratings (and sometimes on demographic information) as the only knowledge sources for generating item proposals for their users. Thus, no additional knowledge – such as information about the available movies and their characteristics – has to be entered and maintained in the system.

Content-based recommendation techniques, use different knowledge sources to make predictions whether a user will like an item. The major knowledge sources exploited by content-based systems include category and genre information, as well as keywords that can often be automatically extracted from textual item descriptions. Similar to CF, a major advantage of content-based recommendation methods is the comparably low cost for knowledge acquisition and maintenance.

Both collaborative and content-based recommender algorithms have their advantages and strengths. However, there are many situations for which these approaches are not the best choice. Typically, we do not buy a house, a car, or a computer very frequently. In such a scenario, a pure CF system will not perform well because of the low number of available ratings. Furthermore, time spans play an important role. For example, five-year-old ratings for computers might be rather inappropriate for content-based recommendation. The same is true for items such as cars or houses, as user preferences evolve over time because of, for example, changes in lifestyles or family situations. Finally, in more complex product domains such as cars, customers often want to define their requirements explicitly – for example, “the maximum price of the car is x and the color should be black”. The formulation of such requirements is not typical for pure collaborative and content-based recommendation frameworks.

Knowledge-based recommender systems help us tackle the aforementioned challenges. The advantage of these systems is that no ramp-up problems exist, because no rating data are needed for the calculation of recommendations. Recommendations are calculated independently of individual user ratings: either in the form of similarities between customer requirements and items or on the basis of explicit recommendation rules. The recommendation process of knowledge-based recommender applications is highly interactive, a foundational property that is a reason for their characterization as conversational systems.

Two basic types of knowledge-based recommender systems are *constraint based and case-based systems* .Both approaches are similar in terms of the recommendation process: the user must specify the requirements, and the system tries to identify a solution. If no solution can be found, the user must change the requirements. The system may also provide explanations for the recommended items. These recommenders differ in the way they use the provided knowledge:

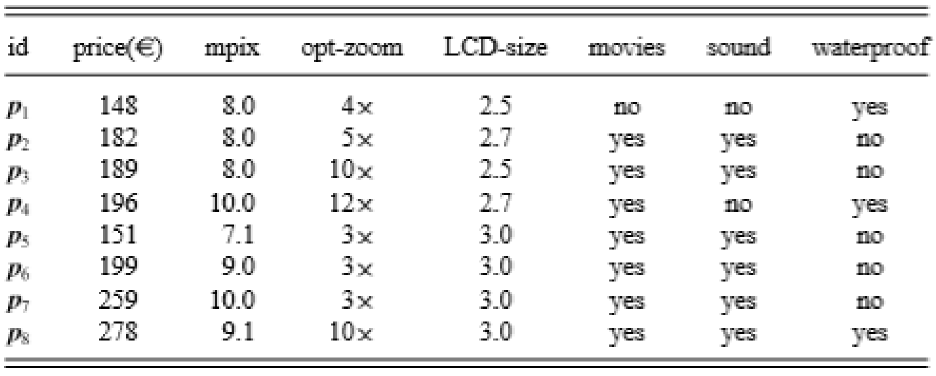
- Case-based recommenders focus on the retrieval of similar items on the basis of different types of similarity measures. It uses similarity metrics to retrieve items that are similar to the specified customer requirements.

- Constraint-based recommenders rely on an explicitly defined set of recommendation rules. Here, the set of recommended items is determined by, for instance, searching for a set of items that fulfill the recommendation rules.

Knowledge representation and reasoning

Knowledge-based systems rely on detailed knowledge about item characteristics. A snapshot of such an item catalog is shown in the following table for the digital camera domain.

Table 4.1. *Example product assortment: digital cameras (Felfernig et al. 2009).*



id	price(€)	mpix	opt-zoom	LCD-size	movies	sound	waterproof
p_1	148	8.0	4×	2.5	no	no	yes
p_2	182	8.0	5×	2.7	yes	yes	no
p_3	189	8.0	10×	2.5	yes	yes	no
p_4	196	10.0	12×	2.7	yes	no	yes
p_5	151	7.1	3×	3.0	yes	yes	no
p_6	199	9.0	3×	3.0	yes	yes	no
p_7	259	10.0	3×	3.0	yes	yes	no
p_8	278	9.1	10×	3.0	yes	yes	yes

The recommendation problem consists of selecting items from this catalog that match the user's needs, preferences, or hard requirements.

We now discuss how the required domain knowledge is encoded in typical knowledge-based recommender systems. A constraint-based recommendation problem can, in general, be represented as a constraint satisfaction problem that can be solved by a constraint solver or in the form of a conjunctive query that is executed and solved by a database engine. Case-based recommendation systems mostly exploit similarity metrics for the retrieval of items from a catalog

Constraints:

A classical constraint satisfaction problem (CSP) can be described by a-tuple (V,D,C) where

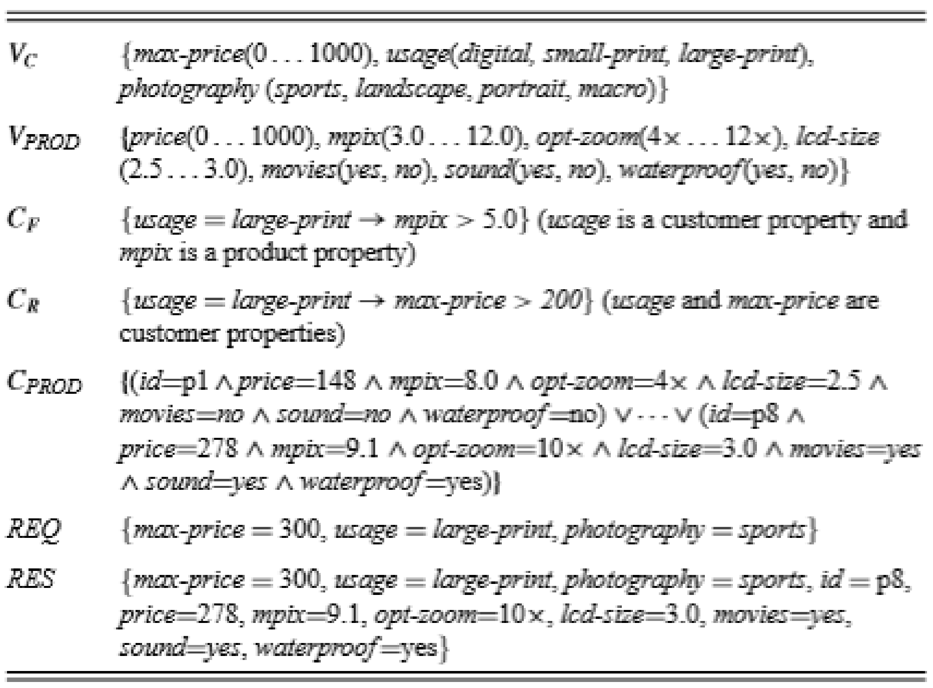
- V is a set of variables
- D is a set of finite domains for these variables

- C is a set of constraints that describes the combinations of values the variables can simultaneously take.

A solution to a CSP corresponds to an assignment of a value to each variable in V in a way that all constraints are satisfied.

Constraint-based recommender systems can build on this formalism and exploit a recommender knowledge base that typically includes two different sets of variables (V = VC ∪ VPROD), one describing potential customer requirements and the other describing product properties. Three different sets of constraints (C = CR ∪ CF ∪ CPROD) define which items should be recommended to a customer in which situation.

Table 4.2. *Example recommendation task (V_C, V_{PROD}, C_R, C_F, C_{PROD}, REQ) and the corresponding recommendation result (RES).*



V_C	{*max-price*(0 . . . 1000), *usage*(*digital*, *small-print*, *large-print*), *photography* (*sports*, *landscape*, *portrait*, *macro*)}
V_{PROD}	{*price*(0 . . . 1000), *mpix*(3.0 . . . 12.0), *opt-zoom*(4× . . . 12×), *lcd-size* (2.5 . . . 3.0), *movies*(*yes*, *no*), *sound*(*yes*, *no*), *waterproof*(*yes*, *no*)}
C_F	{*usage* = *large-print* → *mpix* > 5.0} (*usage* is a customer property and *mpix* is a product property)
C_R	{*usage* = *large-print* → *max-price* > *200*} (*usage* and *max-price* are customer properties)
C_{PROD}	{(*id*=p1 ∧ *price*=148 ∧ *mpix*=8.0 ∧ *opt-zoom*=4× ∧ *lcd-size*=2.5 ∧ *movies*=*no* ∧ *sound*=*no* ∧ *waterproof*=no) ∨ · · · ∨ (*id*=p8 ∧ *price*=278 ∧ *mpix*=9.1 ∧ *opt-zoom*=10× ∧ *lcd-size*=3.0 ∧ *movies*=*yes* ∧ *sound*=*yes* ∧ *waterproof*=yes)}
REQ	{*max-price* = 300, *usage* = *large-print*, *photography* = *sports*}
RES	{*max-price* = 300, *usage* = *large-print*, *photography* = *sports*, *id* = p8, *price*=278, *mpix*=9.1, *opt-zoom*=10×, *lcd-size*=3.0, *movies*=*yes*, *sound*=*yes*, *waterproof*=yes}

- Customer properties (VC) describe the possible customer requirements (see Table 4.2). The customer property max-price denotes the maximum price acceptable for the customer, the property usage denotes the planned usage of photos (print versus digital organization), and photography denotes the predominant type of photos to be taken; categories are, for example, sports or portrait photos.
- Product properties (VPROD) describe the properties of products in an assortment (see Table 4.2); for example, mpix denotes possible resolutions of a digital camera
- Compatibility constraints (CR) define allowed instantiations of customer properties – for example, if large-size photoprints are required, the maximal accepted price must be higher than 200

- Filter conditions (CF) define under which conditions which products should be selected – in other words, filter conditions define the relationships between customer properties and product properties. An example filter condition is large-size photoprints require resolutions greater than 5 mpix.
- Product constraints (CPROD) define the currently available product assortment. An example constraint defining such a product assortment is depicted in Table 4.2. Each conjunction in this constraint completely defines a product (item) – all product properties have a defined value.

Formally, each solution to the CSP ($V = VC \cup VPROD, D, C = CR \cup CF \cup CPROD \cup REQ$) corresponds to a consistent recommendation.

<u>Conjunctive queries:</u>

Here the main task is to construct a conjunctive database query that is executed against the item catalog. A conjunctive query is a database query with a set of selection criteria that are connected conjunctively.

For example, $\sigma[mpix \geq 10, price < 300](P)$ is such a conjunctive query on the database table P, where σ represents the selection operator and $[mpix \geq 10, price < 300]$ the corresponding selection criteria. If we exploit conjunctive queries (database queries) for item selection purposes, VPROD and CPROD are represented by a database table P. Table attributes represent the elements of VPROD and the table entries represent the constraint(s) in CPROD.

<u>Cases and similarities:</u>

In case-based recommendation approaches, items are retrieved using similarity measures that describe to which extent item properties match some given user's requirements. The distance similarity of an item p to the requirements $r \in REQ$ is often defined as shown in the following Formula. In this context, sim(p,r) expresses for each item attribute value $\varphi r(p)$ its distance to the customer requirement $r \in REQ$. Furthermore, wr is the importance weight for requirement r.

$$similarity(p, REQ) = \frac{\sum_{r \in REQ} w_r * sim(p, r)}{\sum_{r \in REQ} w_r}$$

- There are properties a customer would like to maximize("More Is Better" (MIB) properties) – for example, the resolution of a digital camera.

- There are also properties that customers want to minimize("Less Is Better" (LIB)properties)for example, the price of a digital camera or the risk level of a financial service.
- In the case of MIB properties, the local similarity between p and r is calculated as follows:

$$sim(p, r) = \frac{\phi_r(p) - min(r)}{max(r) - min(r)}$$

- The local similarity between p and r in the case of LIB properties is calculated as follows:

$$sim(p, r) = \frac{max(r) - \phi_r(p)}{max(r) - min(r)}$$

- There are situations in which the similarity should be based solely on the distance to the originally defined requirements. For such cases we have to introduce a third type of local similarity function:

$$sim(p, r) = 1 - \frac{|\phi_r(p) - r|}{max(r) - min(r)}$$

<u>Interacting with constraint-based recommenders</u>

The general interaction flow of a knowledge-based, conversational recommender can be summarized as follows.

- The user specifies his or her initial preferences – for example, by using a web-based form. Such forms can be identical for all users or personalized to the specific situation of the current user. Some systems use a question/answer preference elicitation process, in which the questions can be asked either all at once or incrementally in a wizard-style, interactive dialog.
- When enough information about the user's requirements and preferences has been collected, the user is presented with a set of matching items. Optionally, the user can ask for an explanation as to why a certain item was recommended.
- The user might revise his or her requirements, for instance, to see alternative solutions or narrow down the number of matching items.

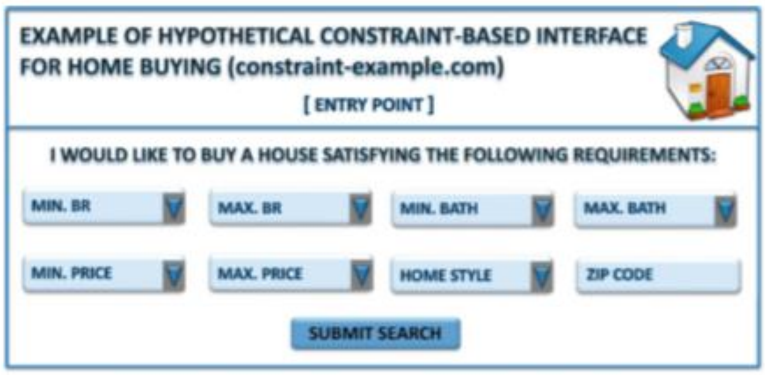

EXAMPLE OF HYPOTHETICAL CONSTRAINT-BASED INTERFACE FOR HOME BUYING (constraint-example.com)
[ENTRY POINT]
I WOULD LIKE TO BUY A HOUSE SATISFYING THE FOLLOWING REQUIREMENTS:
MIN. BR
MAX. BR
MIN. BATH
MAX. BATH
MIN. PRICE
MAX. PRICE
HOME STYLE
ZIP CODE
SUBMIT SEARCH

Figure 5.2: A hypothetical example of an initial user interface for a constraint-based recommender (constraint-example.com)

Different techniques to support users in the interaction with constraint-based recommender applications

These techniques help improve the usability of these applications and achieve higher user acceptance in dimensions such as trust or satisfaction with the recommendation process and the output quality.

1. **Defaults:**
   Proposing default values: Defaults are an important means to support customers in the requirements specification process, especially in situations in which they are unsure about which option to select or simply do not know technical details. Defaults can support customers in choosing a reasonable alternative(an alternative that realistically fits the current preferences). For example, if a customer is interested in printing large-format pictures from digital images, the camera should support a resolution of more than 5.0 megapixels (default). The negative side of the coin is that defaults can also be abused to manipulate consumers to choose certain options.
   Defaults can be specified in various ways:
   - Static defaults: In this case, one default is specified per customer property – for example, default(usage)=large-print, because typically users want to generate posters from high-quality pictures.
   - Dependent defaults: In this case a default is defined on different combinations of potential customer requirements – for example, default(usage=smallprint, max-price)=300.
   - Derived defaults: When the first two default types are strictly based on a declarative approach, this third type exploits existing interaction logs for the automated derivation of default values. The following table shows an example for derived defaults.

Table 4.3. *Example of customer interaction data.*

customer (user)	price	opt-zoom	lcd-size
cu_1	400	10×	3.0
cu_2	300	10×	3.0
cu_3	150	4×	2.5
cu_4	200	5×	2.7
cu_5	200	5×	2.7

The only currently known requirement of a new user should be price=400; the task is to find ult value for the customer requirement on the optical zoom (opt-zoom). From the interaction log we see that there exists a customer (cu1) who had similar requirements (price=400). Thus, we could take cu1's choice for the optical zoom as a default also for the new user. Derived defaults can be determined based on various schemes ;basic example approaches are, 1-*nearest neighbor and weighted majority voter.*

- *1-Nearest neighbor*: It can be used for the prediction of values for one or a set of properties in VC. The basic idea is to determine the entry of the interaction log that is as close as possible to the set of requirements (REQ) specified by the customer.
- *Weighted majority voter*: It proposes customer property values that are based on the voting of a set of neighbor items for a specific property. It operates on a set of n-nearest neighbors, which can be calculated on the basis of Formula

$$similarity(p, REQ) = \frac{\sum_{r \in REQ} w_r * sim(p, r)}{\sum_{r \in REQ} w_r}$$

Selecting the next question.

Besides using defaults to support the user in the requirements specification process, the interaction log and the default mechanism can also be applied for identifying properties that may be interesting for the user within the scope of a recommendation session. For example, if a user has already specified requirements regarding the properties price and opt-zoom, defaults could propose properties that the user could be interested to specify next.

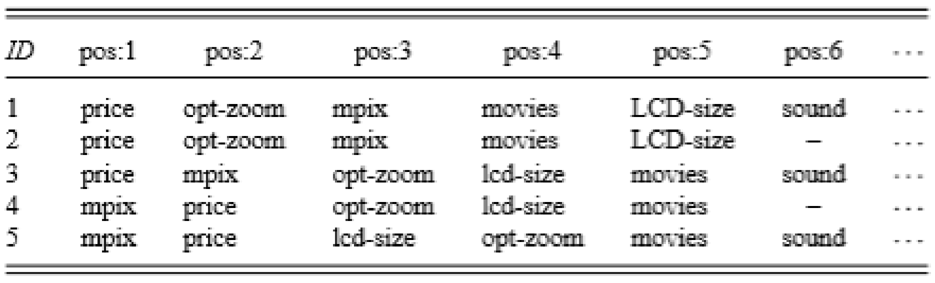
One basic approach to the determination of defaults for the presentation of selectable customer properties recommendation is based on the *principle of frequent usage (popularity*). Such a popularity value can be calculated using the following Formula.

$$popularity(attribute, pos) = \frac{\#selections(attribute, pos)}{\#sessions}$$

Here the recommendation of a question depends strictly on the number of previous selections of other users .

Another approach for supporting question selection is to apply *weighted majority voters*. It considers the preferences of multiple past users (neighbors) with similar requirements to the current user.

Table 4.4. *Order of selected customer properties; for example, in session 4* (ID = 4) mpix *has been selected as first customer property to be specified.*



ID	pos:1	pos:2	pos:3	pos:4	pos:5	pos:6	...
1	price	opt-zoom	mpix	movies	LCD-size	sound	...
2	price	opt-zoom	mpix	movies	LCD-size	–	...
3	price	mpix	opt-zoom	lcd-size	movies	sound	...
4	mpix	price	opt-zoom	lcd-size	movies	–	...
5	mpix	price	lcd-size	opt-zoom	movies	sound	...

Dealing with unsatisfiable requirements and empty result sets:

In many cases, a particular query might return an empty set of results. In other cases, the set of returned results might not be large enough to meet the user requirements. In such cases, a user has two options. If it is deemed that a straightforward way of repairing the constraints does not exist, she may choose to start over from the entry point. Alternatively, she may decide to change or relax the constraints for the next interactive iteration.

In such cases, it is often helpful to provide the user with some guidance on relaxing the current requirements. Such proposals are referred to as repair proposals. The idea is to be able to determine minimal sets of inconsistent constraints, and present them to the user. It is easier for the user to assimilate minimal sets of inconsistent constraints, and find ways of relaxing one or more of the constraints in these sets.

Consider the home-buying example, in which it may be found that the user has specified many requirements, but the only mutually inconsistent pair of requirements is Max-Price < 100,000 and Min-Bedrooms > 5. If this pair of constraints is presented to the user, she can understand that she either needs to increase the maximum price she is willing to pay, or she needs to settle for a smaller number of bedrooms.

Calculating conflict sets.

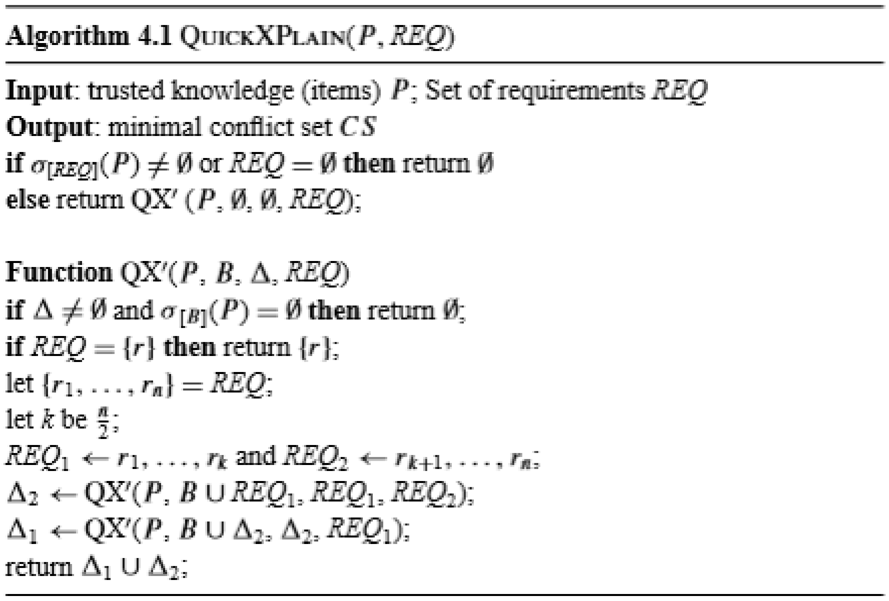
A conflict set (CS) is defined as a subset of the requirements such that no product in the catalog satisfies all of those requirements.

In the context of our problem setting, a diagnosis is a minimal set of user requirements whose repair (adaptation) will allow the retrieval of a recommendation. Given P ={ p1,p2,...,pn} and

REQ ={ r1,r2,...,rm} where σ[REQ](P)=∅, a knowledge-based recommender system would calculate a set of diagnoses ={ d1,d2,...,d k}where σ[REQ−di](P)=∅∀di ∈.A diagnosis is a minimal set of elements{r1,r2,...,r k}=d⊆REQ that have to be repaired in order to restore consistency with the given product assortment so at least one solution can be found: σ[REQ−d](P)=∅.

QuickXPlain is an algorithm that calculates one conflict set at a time for a given set of constraints. Its divide-and-conquer strategy helps to significantly accelerate the performance compared to other approaches.

It divides the set of requirements into the subsets REQ1 and REQ2. If both subsets contain about 50 percent of the requirements (the splitting factor is n 2), all the requirements contained in REQ2 can be deleted (ignored) after a single consistency check if σ[REQ1](P) =∅. The splitting factor of n 2 is generally recommended; however, other factors can be defined. In the best case (e.g., all elements of the conflict belong to subset REQ1) the algorithm requires log2 n/ u +2u consistency checks; in the worst case, the number of consistency checks is 2u(log2 n/ u +1), u + 1), where u is the number of elements contained in the conflict set.



```
Algorithm 4.1 QUICKXPLAIN($P, REQ$)
Input: trusted knowledge (items) $P$; Set of requirements $REQ$
Output: minimal conflict set $CS$
if $\sigma_{[REQ]}(P) \neq \emptyset$ or $REQ = \emptyset$ then return $\emptyset$
else return QX' ($P, \emptyset, \emptyset, REQ$);

Function QX'($P, B, \Delta, REQ$)
if $\Delta \neq \emptyset$ and $\sigma_{[B]}(P) = \emptyset$ then return $\emptyset$;
if $REQ = \{r\}$ then return $\{r\}$;
let $\{r_1, \dots, r_n\} = REQ$;
let $k$ be $\frac{n}{2}$;
$REQ_1 \leftarrow r_1, \dots, r_k$ and $REQ_2 \leftarrow r_{k+1}, \dots, r_n$;
$\Delta_2 \leftarrow$ QX'($P, B \cup REQ_1, REQ_1, REQ_2$);
$\Delta_1 \leftarrow$ QX'($P, B \cup \Delta_2, \Delta_2, REQ_1$);
return $\Delta_1 \cup \Delta_2$;
```

Diagnoses can be calculated by resolving conflicts in the given set of requirements. Because of its minimality, one conflict can be easily resolved by deleting one of the elements from the conflict set. After having deleted at least one element from each of the identified conflict sets, we are able to present a corresponding diagnosis.

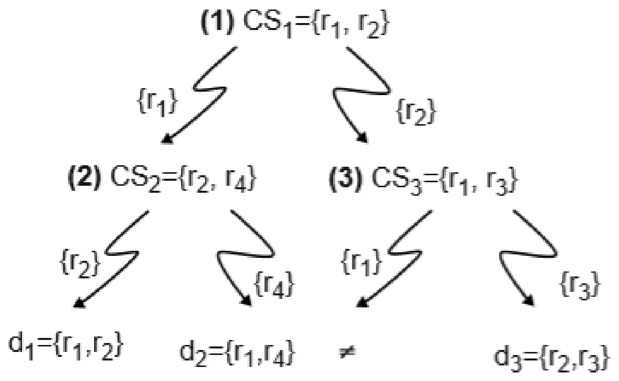


Figure 4.1. Calculating diagnoses for unsatisfiable requirements.

Ranking the items/utility-based recommendation

It is important to rank recommended items according to their utility for the customer. Because of primacy effects that induce customers to preferably look at and select items at the beginning of a list, such rankings can significantly increase the trust in the recommender application as well as the willingness to buy.

In knowledge-based conversational recommenders, the ranking of items can be based on the multi-attribute utility theory (MAUT), which evaluates each item with regard to its utility for the customer. Each item is evaluated according to a predefined set of dimensions that provide an aggregated view on the basic item properties. For example, quality and economy are dimensions in the domain of digital cameras; availability, risk, and profit are such dimensions in the financial services domain. The following table exemplifies the definition of scoring rules that define the relationship between item properties and dimensions. For example, a digital camera with a price lower than or equal to 250 is evaluated, with Q score of 5 regarding the dimension quality and 10 regarding the dimension economy.

Table 4.7. *Example scoring rules regarding the dimensions* quality *and* economy.

	value	quality	economy
price	≤250	5	10
	>250	10	5
mpix	≤8	4	10
	>8	10	6
opt-zoom	≤9	6	9
	>9	10	6
LCD-size	≤2.7	6	10
	>2.7	9	5
movies	yes	10	7
	no	3	10
sound	yes	10	8
	no	7	10
waterproof	yes	10	6
	no	8	10

We can determine the utility of each item p in P for a specific customer . The customer-specific item utility is calculated on the basis of following Formula .

$$utility(p) = \sum_{j=1}^{\#(dimensions)} interest(j) * contribution(p, j)$$

Here, the index j iterates over the number of predefined dimensions (in our example, #(dimensions)=2: quality and economy), interest(j) denotes a user’s interest in dimension j, and contribution(p,j) denotes the contribution of item p to the interest dimension j. The value for contribution(p,j) can be calculated by the scoring rules defined in Table 4.7 – for example, the contribution of item p1 to the dimension quality is 5+4+6+ 6+3+7+10 = 41, whereas its contribution to the dimension economy is 10+10+9+10+10+10+6 = 65.

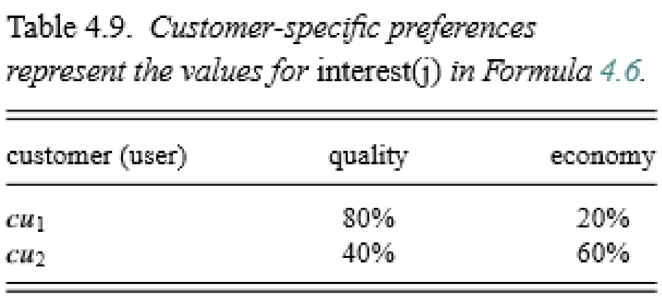


Table 4.9. *Customer-specific preferences represent the values for* interest(j) *in Formula 4.6.*

customer (user)	quality	economy
cu_1	80%	20%
cu_2	40%	60%

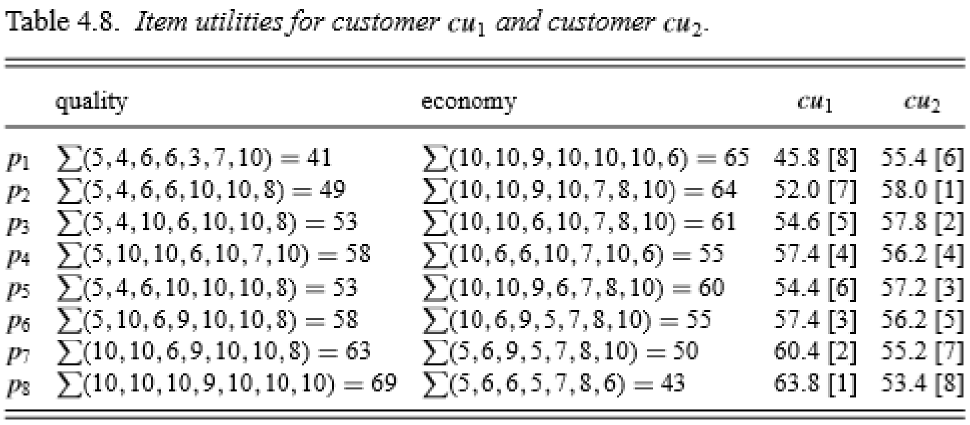


Table 4.8. *Item utilities for customer cu_1 and customer cu_2.*

	quality	economy	cu_1	cu_2
p_1	$\sum(5,4,6,6,3,7,10) = 41$	$\sum(10,10,9,10,10,10,6) = 65$	45.8 [8]	55.4 [6]
p_2	$\sum(5,4,6,6,10,10,8) = 49$	$\sum(10,10,9,10,7,8,10) = 64$	52.0 [7]	58.0 [1]
p_3	$\sum(5,4,10,6,10,10,8) = 53$	$\sum(10,10,6,10,7,8,10) = 61$	54.6 [5]	57.8 [2]
p_4	$\sum(5,10,10,6,10,7,10) = 58$	$\sum(10,6,6,10,7,10,6) = 55$	57.4 [4]	56.2 [4]
p_5	$\sum(5,4,6,10,10,10,8) = 53$	$\sum(10,10,9,6,7,8,10) = 60$	54.4 [6]	57.2 [3]
p_6	$\sum(5,10,6,9,10,10,8) = 58$	$\sum(10,6,9,5,7,8,10) = 55$	57.4 [3]	56.2 [5]
p_7	$\sum(10,10,6,9,10,10,8) = 63$	$\sum(5,6,9,5,7,8,10) = 50$	60.4 [2]	55.2 [7]
p_8	$\sum(10,10,10,9,10,10,10) = 69$	$\sum(5,6,6,5,7,8,6) = 43$	63.8 [1]	53.4 [8]

To determine the overall utility of item p1 for a specific customer, we must take into account the customer-specific interest in each of the given dimensions – interest(j). For customer cu1, the utility of item p2 is 49∗0.8+64∗0.2 = 52.0 and the overall utility of item p8 would be 69∗0.8+43∗0.2 = 63.8. For customer cu2, item p2 has the utility 49∗0.4+64∗0.6 = 58.0 and item p8 has the utility 69∗0.4+43∗0.6 = 53.4. Consequently, item p8 has a higher utility (and the highest utility) for cu1, whereas item p2 has a higher utility (and the highest utility) for cu2.

Customer preferences can be explicitly defined by the user (user-defined preferences). Preferences can also be predefined in the form of scoring rules (utility-based preferences).

- *User-defined preferences*:
  The first and most straightforward approach is to directly ask the customer for his or her preferences within the scope of a recommendation session.

- *Utility-based preferences*:
  A second possible approach to determining customer preferences is to apply the scoring rules. For example, that a customer has specified the requirements REQ={ r1 : price <= 200,r2 : mpix = 8.0,r3 : opt-zoom = 10×,r4 : lcd-size <= 2.7}. The dimension quality would be 5+4+10+6 = 25 and the dimension economy would be 10+10+6+10 = 36. This would result in a relative importance for quality with a value of 25/(25+36) = 0.41 and a relative importance for economy with the value 36/(25+36)=0.59.

Interacting with case-based recommenders

Earlier versions of case-based recommenders followed a pure query-based approach, in which users had to specify (and often respecify) their requirements until a target item (an item that fits the user's wishes and needs) has been identified. Especially for nonexperts in the product domain, this type of requirement elicitation process can lead to tedious recommendation sessions, as the interdependent properties of items require a substantial domain knowledge to perform well.

The drawback of pure query-based approaches motivated the development of browsing-based approaches to item retrieval, in which users – maybe not knowing what they are seeking – are navigating in the item space with the goal to find useful alternatives. Critiquing is an effective way to support such navigations and, in the meantime, it is one of the key concepts of case-based recommendation

Critiquing

The idea of critiquing is that users specify their change requests in the form of goals that are not satisfied by the item currently under consideration (entry item or recommended item). If, for example, the price of the currently displayed digital camera is too high, a critique cheaper can be activated; if the user wants to have a camera with a higher resolution (mpix), a corresponding critique more mpix can be selected.

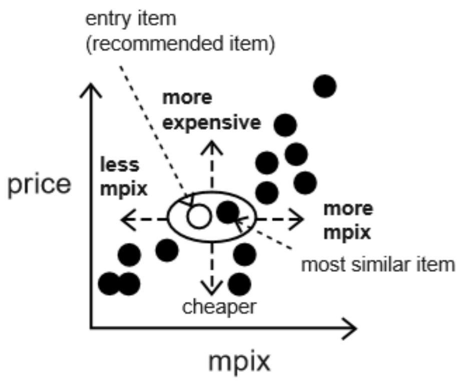
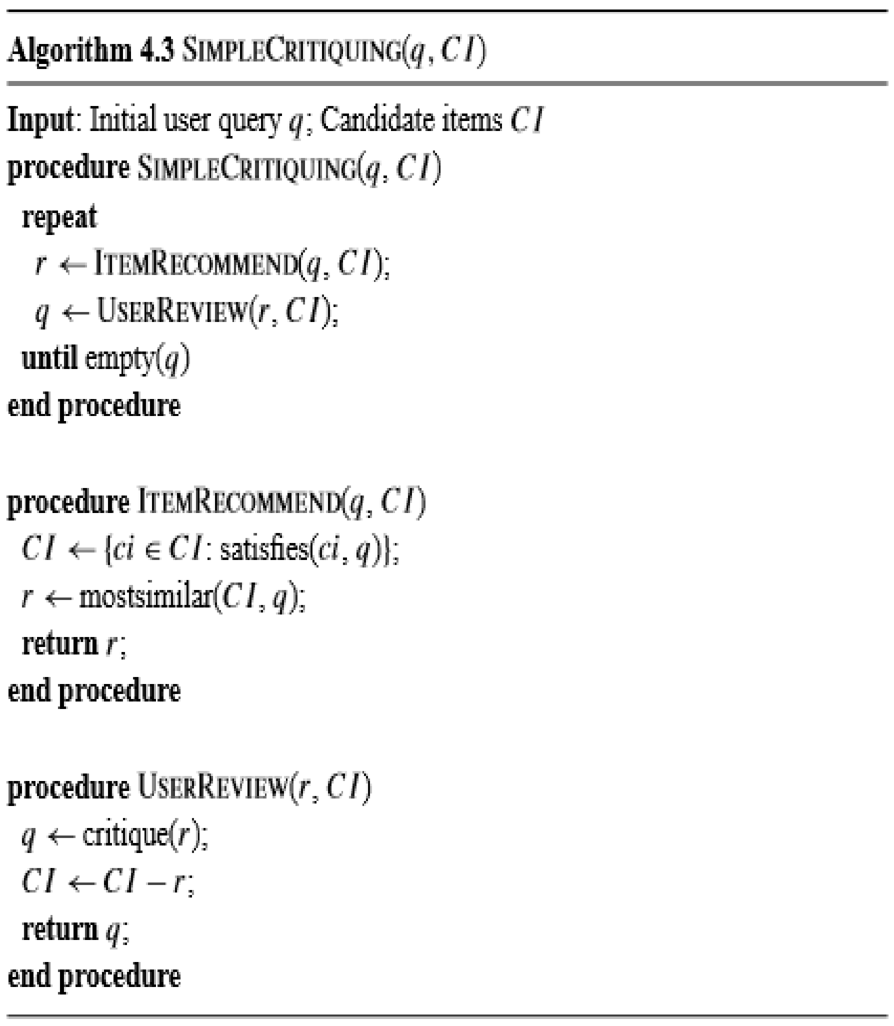


Figure 4.3. Critique-based navigation: items recommended to the user can be critiqued regarding different item properties (e.g., *price* or *mpix*).

The goal of critiquing is to achieve time savings in the item selection process and, at the same time, achieve at least the same recommendation quality as standard query-based approaches. The major steps of a critiquing-based recommender application are the following (see Algorithm 4.3, SimpleCritiquing).



Algorithm 4.3 SimpleCritiquing(q, CI)

```
Input: Initial user query $q$; Candidate items $CI$
procedure SimpleCritiquing($q, CI$)
  repeat
    $r \leftarrow$ ItemRecommend($q, CI$);
    $q \leftarrow$ UserReview($r, CI$);
  until empty($q$)
end procedure

procedure ItemRecommend($q, CI$)
  $CI \leftarrow \{ci \in CI$: satisfies($ci, q$)$\}$;
  $r \leftarrow$ mostsimilar($CI, q$);
  return $r$;
end procedure

procedure UserReview($r, CI$)
  $q \leftarrow$ critique($r$);
  $CI \leftarrow CI - r$;
  return $q$;
end procedure
```

- *Item recommendation*.
  The inputs for the algorithm SimpleCritiquing4 are an initial user query q, which specifies an initial set of requirements, and a set of candidate items CI that initially consists of all the available items (the product assortment). The algorithm first activates the procedure ItemRecommend, which is responsible for selecting an item r to be presented to the user. We denote the item that is displayed in the first critiquing cycle as entry item and all other items displayed thereafter as recommended items. In the first critiquing cycle, the retrieval of such items is based on a user query q that represents a set of initial requirements. Entry items are typically determined by calculating the similarity between the requirements and the candidate items. After the first critiquing cycle has been completed, recommended items are determined by the procedure ItemRecommend on the basis of the similarity between the currently recommended item and those items that fulfill the criteria of the critique specified by the user.

- *Item reviewing*.

The user reviews the recommended (entry) item and either accepts the recommendation or selects another critique, which triggers a new critiquing cycle (procedureUserReview). If a critique has been triggered, only the items (the candidate items) that fulfill the criteria defined in the critique are further taken into account – this reduction of CI is done in procedure ItemRecommend. For example, if a user activates the critique cheaper, and the price of the recommended (entry) camera is 300, the recommender excludes cameras with a price greater than or equal to 300 in the following critiquing cycle.

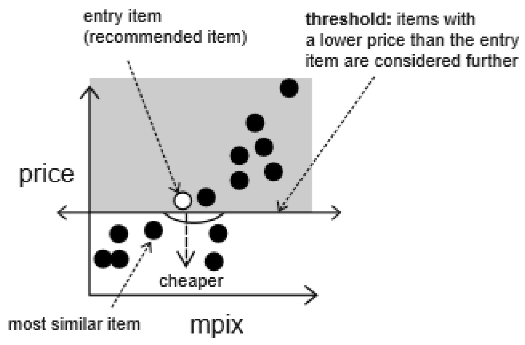


Figure 4.4. Critique-based navigation: remaining candidate items (items with bright background) after a critique on the entry item property *price*.

Compound critiquing

Unit critiques or simple critiquing allow the definition of change requests that are related to a single item property. Unit critiques have a limited capability to effectively narrow down the search space. For example, the unit critique on price in Figure eliminates only about half the items.

Allowing the specification of critiques that operate over multiple properties can significantly improve the efficiency of recommendation dialogs, for example, in terms of a reduced number of critiquing cycles. Such critiques are denoted as compound critiques. The effect of compound critiques on the number of eliminated items (items not fulfilling the criteria of the critique) is shown in the following Figure. The compound critique cheaper and more mpix defines additional goals on two properties that should be fulfilled by the next proposed recommendation.

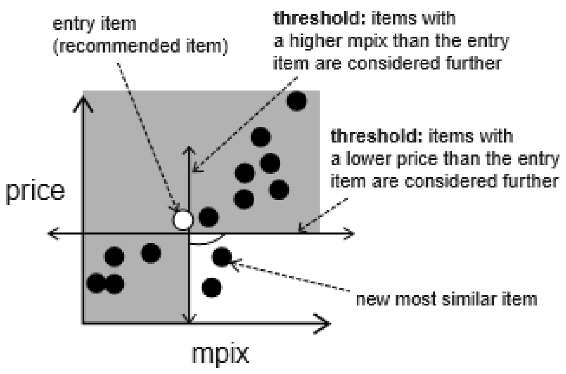

entry item
(recommended item)
threshold: items with a higher mpix than the entry item are considered further
threshold: items with a lower price than the entry item are considered further
price
new most similar item
mpix

Figure 4.5. Critique-based navigation: remaining candidate items (items with bright background) after a compound critique on *price* and *mpix*.

An important advantage of compound critiques is that they allow a faster progression through the item space. However, compound critiques still have disadvantages as long as they are formulated statically, as all critique alternatives are available for every item displayed.

Dynamic critiquing

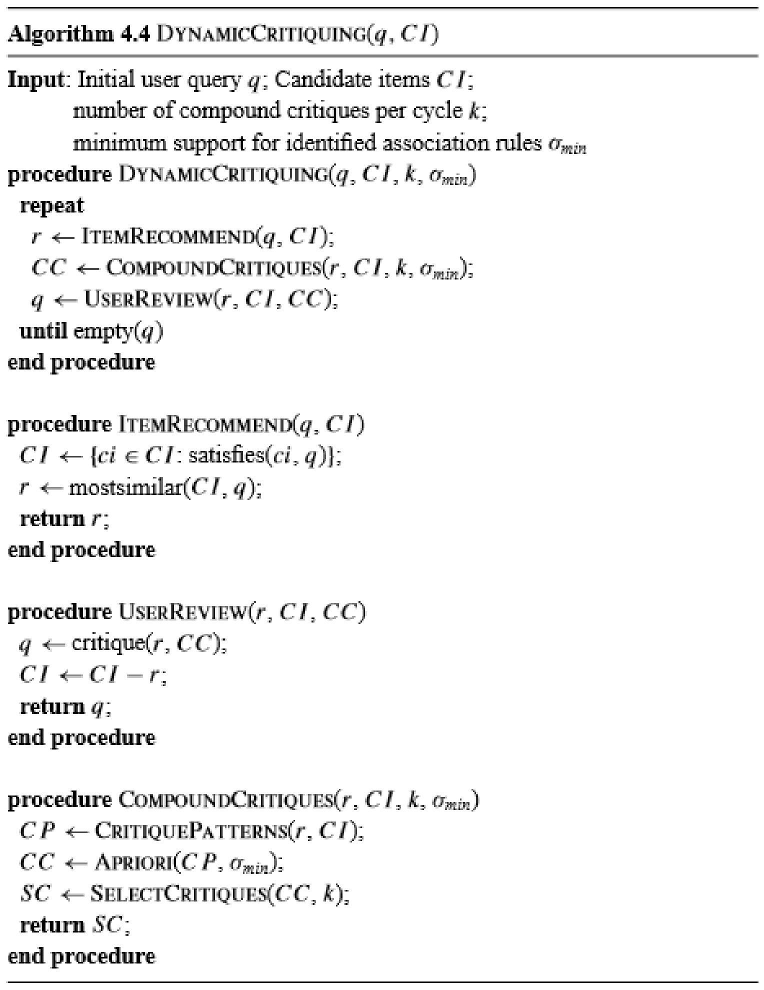
Dynamic critiquing exploits patterns, which are generic descriptions of differences between the recommended (entry) item and the candidate items – these patterns are used for the derivation of compound critiques. Critiques are denoted as dynamic because they are derived on the fly in each critiquing cycle. Dynamic critiques are calculated using the concept of association rule mining.

A dynamic critiquing cycle consists of the following basic steps:

The inputs for the algorithm are an initial user query q, which specifies the initial set of requirements, a set of candidate items CI that initially consists of all the available items, k as the maximum number of compound critiques to be shown to the user in one critiquing cycle, and σmin as the minimum support value for calculated association rules.

- *Item recommendation*.:
  DynamicCritiquing algorithm first activates the procedure ItemRecommend, which is responsible for returning one recommended item r (respectively, entry item in the first critiquing cycle). On the basis of this item, the algorithm starts the calculation of compound critiques cci ∈ CC by activating the procedure Compound Critiques, which itself activates the procedures CritiquePatterns (identification of critique patterns), APriori (mining compound critiques from critique patterns), and SelectCritiques (ranking compound critiques).

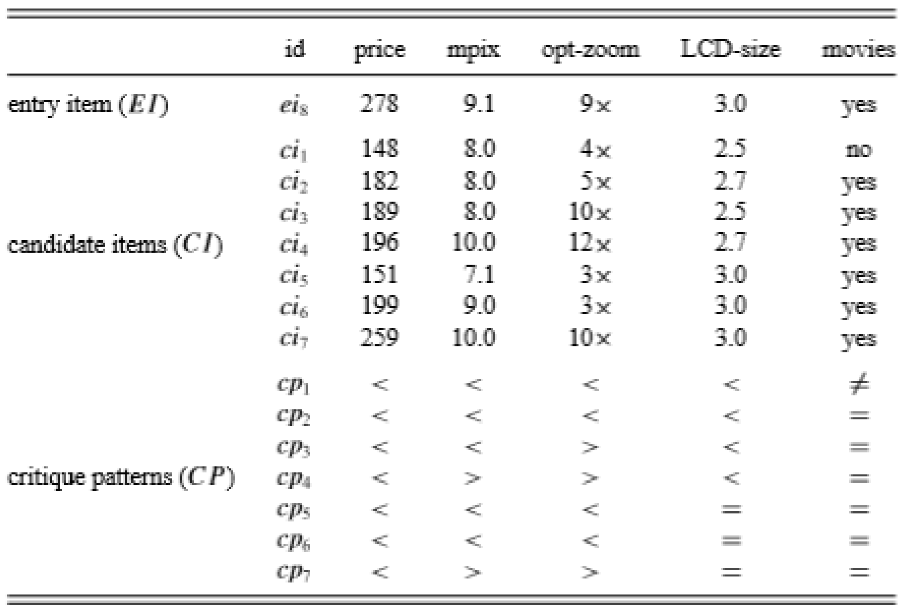
The identified compound critiques in CC are then shown to the user inUserReview. If the user selects a critique – which could be a unit critique on a specific item property as well as a compound critique – this forms the criterion of the new user query q. If the resulting query q is empty, the critiquing cycle can be stopped.



```
Algorithm 4.4 DynamicCritiquing($q$, $CI$)
Input: Initial user query $q$; Candidate items $CI$;
       number of compound critiques per cycle $k$;
       minimum support for identified association rules $\sigma_{min}$
procedure DynamicCritiquing($q$, $CI$, $k$, $\sigma_{min}$)
  repeat
    $r \leftarrow$ ItemRecommend($q$, $CI$);
    $CC \leftarrow$ CompoundCritiques($r$, $CI$, $k$, $\sigma_{min}$);
    $q \leftarrow$ UserReview($r$, $CI$, $CC$);
  until empty($q$)
end procedure

procedure ItemRecommend($q$, $CI$)
  $CI \leftarrow \{ci \in CI$: satisfies($ci$, $q$)$\}$;
  $r \leftarrow$ mostsimilar($CI$, $q$);
  return $r$;
end procedure

procedure UserReview($r$, $CI$, $CC$)
  $q \leftarrow$ critique($r$, $CC$);
  $CI \leftarrow CI - r$;
  return $q$;
end procedure

procedure CompoundCritiques($r$, $CI$, $k$, $\sigma_{min}$)
  $CP \leftarrow$ CritiquePatterns($r$, $CI$);
  $CC \leftarrow$ Apriori($CP$, $\sigma_{min}$);
  $SC \leftarrow$ SelectCritiques($CC$, $k$);
  return $SC$;
end procedure
```

- *Identification of critique patterns*. Critique patterns are a generic representation of the differences between the currently recommended item (entry item) and the candidate items. Table 4.12 depicts a simple example for the derivation of critique patterns, where item ei8 is assumed to be the entry item and the items {ci1,...,ci7} are the candidate items. On the basis of this example, critique patterns can be easily generated by comparing the properties of item ei8 with the properties of{ci1,...,ci7}. For example, compared with item ei8, item ci1 is cheaper, has less mpix, a loweropt-zoom, a smaller lcd-size, and does not have a movie functionality. The corresponding critique pattern for item ci1 is (<,<,<,<,=).These patterns are the result of calculating the type of difference for each combination of recommended (entry) and candidate item. In the algorithm DynamicCritiquing, critique patterns are determined on the basis of the procedure CritiquingPatterns.

Table 4.12. *Critique patterns (CP) are generated by analyzing the differences between the* recommended item *(the* entry item, *EI) and the* candidate items *(CI). In this example, item ei_8 is assumed to be the* entry item, *$\{ci_1, \ldots, ci_7\}$ are assumed to be the candidate items, and $\{cp_1, \ldots, cp_7\}$ are the critique patterns.*



	id	price	mpix	opt-zoom	LCD-size	movies
entry item (*EI*)	ei_8	278	9.1	9×	3.0	yes
	ci_1	148	8.0	4×	2.5	no
	ci_2	182	8.0	5×	2.7	yes
	ci_3	189	8.0	10×	2.5	yes
candidate items (*CI*)	ci_4	196	10.0	12×	2.7	yes
	ci_5	151	7.1	3×	3.0	yes
	ci_6	199	9.0	3×	3.0	yes
	ci_7	259	10.0	10×	3.0	yes
	cp_1	$<$	$<$	$<$	$<$	$\neq$
	cp_2	$<$	$<$	$<$	$<$	$=$
	cp_3	$<$	$<$	$>$	$<$	$=$
critique patterns (*CP*)	cp_4	$<$	$>$	$>$	$<$	$=$
	cp_5	$<$	$<$	$<$	$=$	$=$
	cp_6	$<$	$<$	$<$	$=$	$=$
	cp_7	$<$	$>$	$>$	$=$	$=$

- *Mining compound critiques from critique patterns*.
  The next step is to identify compound critiques that frequently co-occur in the set of critique patterns. For critique calculation, the Apriori algorithm is used. The output of this algorithm is a set of association rules p⇒q, which describe relationships between elements in the set of critique patterns. An example is >zoom ⇒<price, which can be derived from the critique patterns of Table 4.12. This rule denotes the fact that given >zoom as part of a critique pattern, <price is contained in the same critique pattern.

Ranking of compound critiques

The number of compound critiques can become very large, which makes it important to filter out the most relevant critiques for the user in each critiquing cycle. Critiques with low support have the advantage of significantly reducing the set of candidate items, but at the same time they decrease the probability of identifying the target item. Critiques with high support can significantly increase the probability of finding the target item. However, these critiques eliminate a low number of candidate cases, which leads to a larger number of critiquing cycles in recommendation sessions. Many existing recommendation approaches rank compound critiques according to the support values of association rules, because the lower the support of the corresponding association rules, the more candidate items can be eliminated. In the algorithm DynamicCritiquing, compound critiques are selected on the basis of the procedure SelectCritiques.

Item reviewing.

At this stage of a recommendation cycle all the relevant information for deciding about the next action is available for the user: the recommended (entry) item and the corresponding set of compound critiques. The user reviews the recommended item and either accepts the recommendation or selects a critique (unit or compound), in which case a new critiquing cycle is started. In the algorithm DynamicCritiquing, item reviews are conducted by the user in UserReview.