The Wumpus World agent is a classic example of a knowledge-based agent used in Artificial Intelligence (AI). It's placed in a simulated cave environment with the following challenges:

Navigating the Cave: The cave is a grid of rooms, and the agent needs to explore it to find its way around.
Avoiding Hazards: There are two main dangers: pits (where the agent can fall in and get stuck) and a monster called the Wumpus (which will eat the agent).
Finding the Goal: The ultimate objective is to locate the gold hidden in one of the rooms and safely return to the starting point.
The Wumpus World agent is limited in its perception. It can only sense its immediate surroundings using:

Knowledge based agent

Wumpus world agent

Breeze: A breeze indicates a pit in an adjacent room.
Stench: A stench signifies the Wumpus lurks in a neighboring room.
Using these limited clues, the agent must build up a knowledge base about the cave and reason about its actions. Here's what the agent does:

Perceive its surroundings: It detects breezes and stenches to infer the location of pits and the Wumpus (although not the exact room).
Update its knowledge base: Based on its perceptions, the agent builds a mental map of the cave, marking safe and unsafe areas.
Make decisions: The agent strategically decides on actions like moving forward, turning, or shooting its single arrow (if it encounters the Wumpus's stench).
The Wumpus World serves as a valuable testbed for studying how AI agents can handle:

Knowledge Representation: The agent needs to represent its knowledge about the cave in a way that facilitates reasoning.
Reasoning: The agent must use logic and deduction to infer the locations of hazards and the gold based on limited information.
Planning: The agent strategizes its movements to explore the cave safely and achieve its goal.
While the Wumpus World itself is a simplified environment, it offers valuable insights into building intelligent agents that can navigate uncertain situations and make informed decisions based on incomplete information.

- **Meta Knowledge:** It is the information/knowledge about knowledge.
- **Heuristic Knowledge:** It is the knowledge regarding a specific topic.
- **Procedural Knowledge:** It gives information about achieving something.
- **Declarative Knowledge:** It is the information which describes a particular object and its attributes.
- **Structural Knowledge:** It describes the knowledge between the objects.

applying scientific principles to build a model or analyzing historical events to understand current world issues.

What is Knowledge?

- Knowledge is **what I know** and Information is **what we know**
- Knowledge can be considered as the distillation of information that has been **collected, classified, organized, integrated, abstracted and value added.**
- Intelligent behavior is not dependent so much on the methods of reasoning as on the knowledge one has to reason with.

while reasoning is a crucial skill for intelligent behavior, it's the knowledge you possess that truly fuels its effectiveness.

Characteristics of Knowledge

- –Knowledge is **huge** (Large in number or quantity).
- –Knowledge is **hard to characterize** accurately.
- –Knowledge differs from data in that it is organized such that it corresponds to the ways it will be used.
- –Knowledge is **interpreted differently** by **different people.**

Knowledge

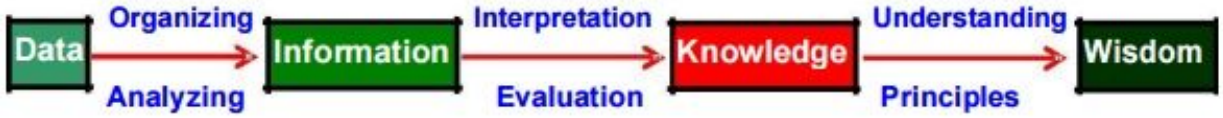


Fig 1 Knowledge Progression

- **Data** : Collection of **disconnected facts**
- **Information** :Emerges when **Relationships between facts** are established and understood. Provides answer to **Who, what, where and when.**
- **Knowledge** : emerges when **relationship between patterns** are identified and understood. Provides answer to **"How"**
- **Wisdom** : is the pinnacle of understanding, uncovers the **principles of relationships that describe patterns**. Provides answer as **" Why"**

Knowledge Model

- **Data and information** : Past data
- **Knowledge :** Present/ enable us to perform
- **Wisdom :** Future – acquire vision for what will be

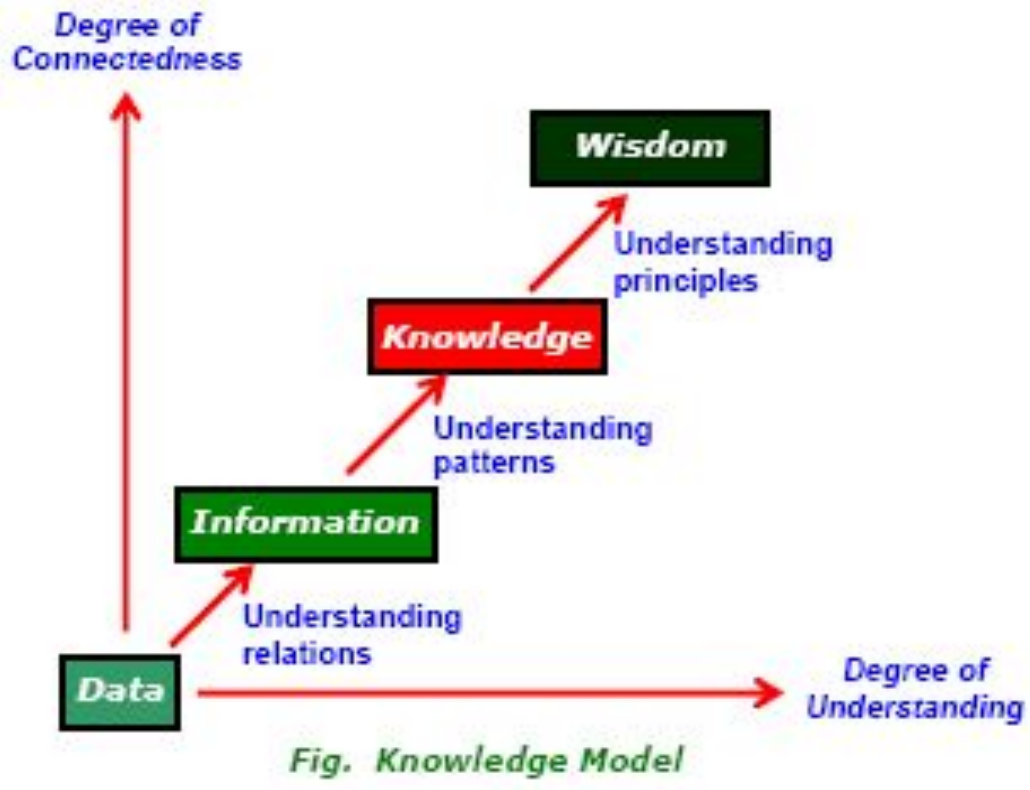


Fig. Knowledge Model

Knowledge based agents / Intelligent agents

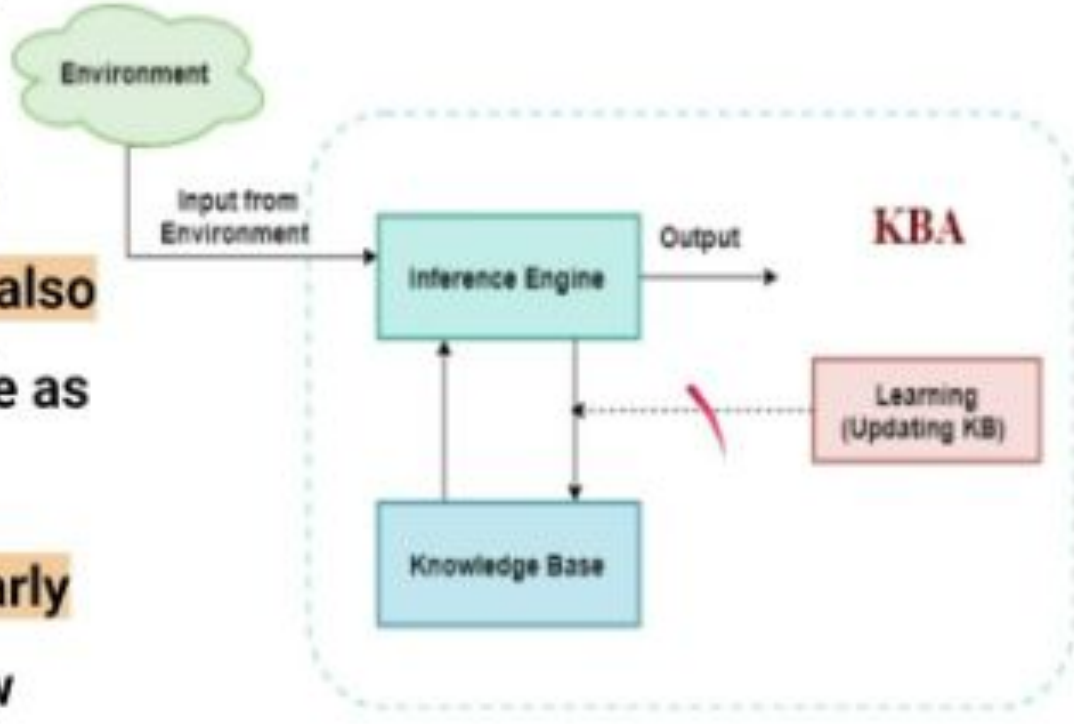
Importance of Knowledge representation in Intelligent Agents

- **Intelligent agents** should have capacity for
  - **Perceiving** : acquiring information from environment
  - **Knowledge Representation** : representing its understanding of the world
  - **Reasoning** : inferring the implications of what it knows and of the choices it has
  - **Acting** : choosing what it want to do and carry it out.

reasoning is a powerful tool that allows us to navigate the world effectively by using our knowledge to make sense of situations and choose the best course of action.

Architecture- Knowledge Based Agents(KBA)

- knowledge-based agent (KBA) take input from the environment by perceiving the environment
- input is taken by the inference engine of the agent and which also communicate with KB to decide as per the knowledge store in KB
- learning element of KBA regularly updates the KB by learning new



Knowledge-based recommender systems

Knowledge-based recommender systems are well suited to the recommendation of items that are not bought on a regular basis. Furthermore, in such item domains, users are generally more active in being explicit about their requirements. A user may often be willing to accept a movie recommendation without much input, but she would be unwilling to accept recommendations about a house or a car without having detailed information about the specific features of the item. Therefore, knowledge-based recommender systems are suited to types of item domains different from those of collaborative and content-based systems.

knowledge-based recommender systems are appropriate in the following situations

1. Customers want to explicitly specify their requirements. Therefore, interactivity is a crucial component of such systems. Note that collaborative and content-based systems do not allow this type of detailed feedback.
2. It is difficult to obtain ratings for a specific type of item because of the greater complexity of the product domain in terms of the types of items and options available.
3. In some domains, such as computers, the ratings may be time-sensitive. The ratings on an old car or computer are not very useful for recommendations because they evolve with changing product availability and corresponding user requirements.

Table 5.1: The conceptual goals of various recommender systems

Approach	**Conceptual Goal**	**Input**
Collaborative	Give me recommendations based on a collaborative approach that leverages the ratings and actions of my peers/myself.	User ratings + community ratings
Content-based	Give me recommendations based on the content (attributes) I have favored in my past ratings and actions.	User ratings + item attributes
Knowledge-based	Give me recommendations based on my explicit specification of the kind of content (attributes) I want.	User specification + item attributes + domain knowledge

There are two primary types of knowledge-based recommender systems

1. Constraint-based recommender systems:

users typically specify requirements or constraints (e.g., lower or upper limits) on the item

Furthermore, domain-specific rules are used to match the user requirements or attributes to item attributes. These rules represent the domain-specific knowledge used by the system. Such rules could take the form of domain-specific constraints on the item attributes (e.g., “Cars before year 1970 do not have cruise control.”). Furthermore, constraint-based systems often create rules relating user attributes to item attributes (e.g., “Older investors do not invest in ultrahigh-risk products.”).

2. Case-based recommender systems:

In case-based recommender systems [102, 116, 377, 558], specific cases are specified by the user as targets or anchor points. Similarity metrics are defined on the item attributes to retrieve similar items to these targets. The similarity metrics are often carefully defined in a domain-specific way. Therefore, the similarity metrics form the domain knowledge that is used in such systems. The returned results are often used as new target cases with some interactive modifications by the user. For example, when a user sees a returned result that is almost similar to what she wants, she might re-issue a query with that target, but with some of the attributes changed to her liking. Alternatively, a directional critique may be specified to prune items with specific attribute values greater (or less) than that of a specific item of interest. A

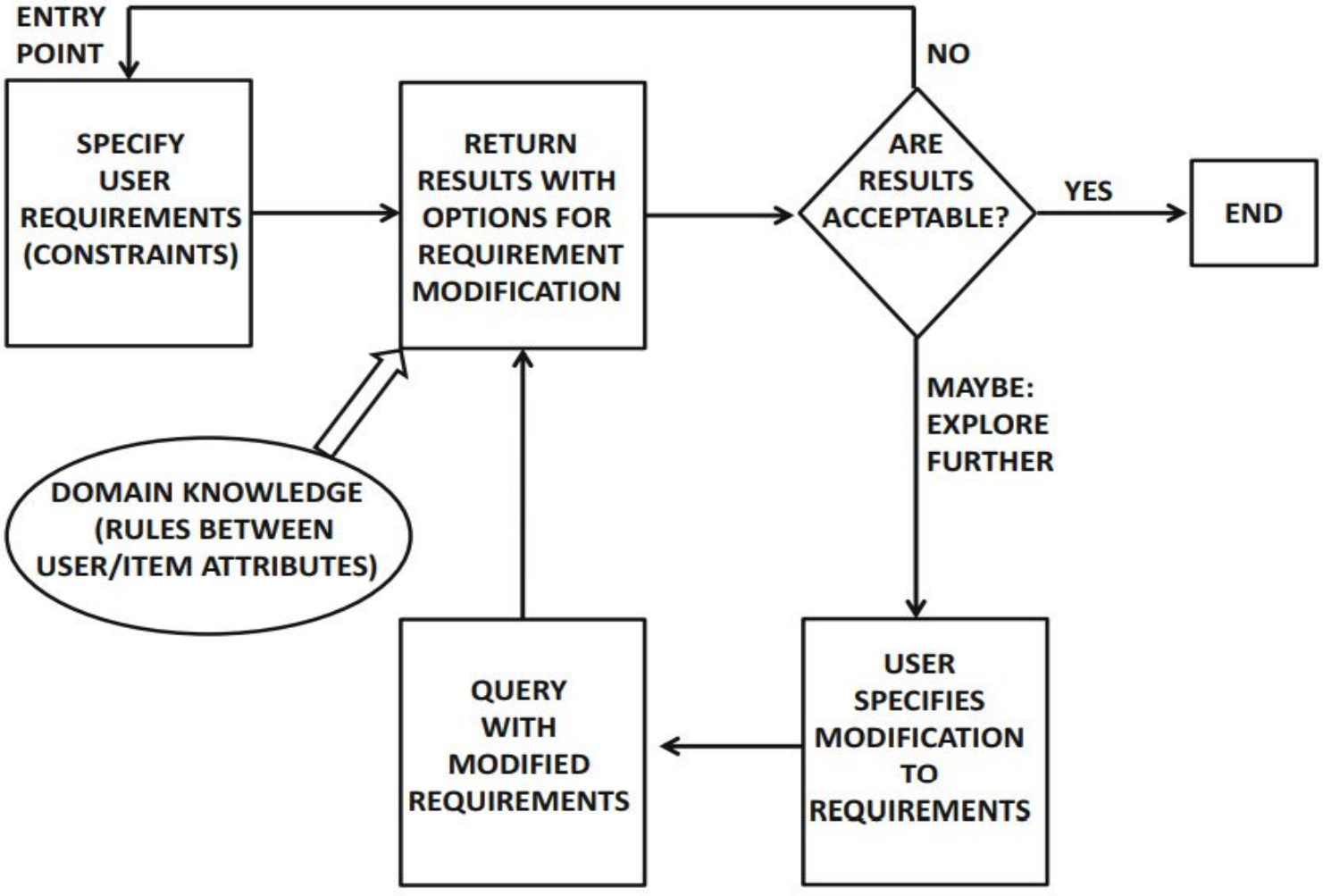
The interaction between user and recommender may take the form

1. Conversational systems: In this case, the user preferences are determined in the context of a feedback loop. The main reason for this is that the item domain is complex, and the user preferences can be determined only in the context of an iterative conversational system.

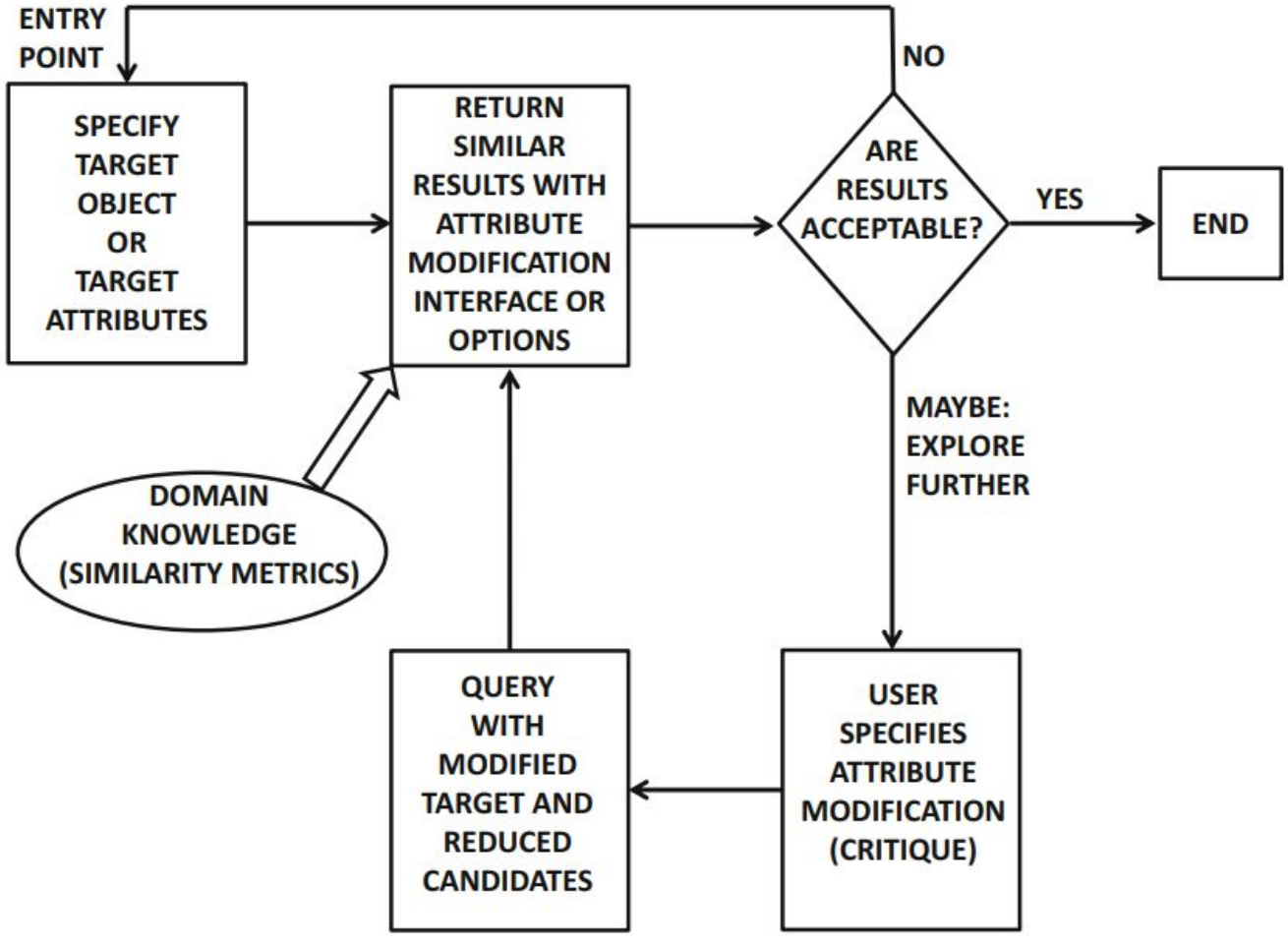
eg. chatgpt can recommend how to plan a trip given the context

2. Search-based systems: In search-based systems, user preferences are elicited by using a preset sequence of questions such as the following: "Do you prefer a house in a suburban area or within the city?"

3. Navigation-based recommendation: In navigation-based recommendation, the user specifies a number of change requests to the item being currently recommended. Through an iterative set of change requests, it is possible to arrive at a desirable item. An example of a change request specified by the user, when a specific house is being recommended is as follows: "I would like a similar house about 5 miles west of the currently recommended house." Such recommender systems are also referred to as critiquing recommender systems



(a) Constraint-based interaction



(b) Case-based interaction

Figure 5.1: Overview of interactive process in knowledge-based recommenders

Constraint-Based Recommender Systems

Constraint-based recommender systems allow the users to specify hard requirements or constraints on the item attributes. Furthermore, a set of rules is used in order to match the customer requirements with item attributes. However, the customers may not always specify their queries in terms of the same attributes that describe the items. Therefore, an additional set of rules is required that relates the customer requirements with the product attributes.

Table 5.2: Examples of attributes in a recommendation application for buying homes

Item-Id	Beds.	Baths.	Locality	Type	Floor Area	Price
1	3	2	Bronx	Townhouse	1600	220,000
2	5	2.5	Chappaqua	Split-level	3600	973,000
3	4	2	Yorktown	Ranch	2600	630,000
4	2	1.5	Yorktown	Condo	1500	220,000
5	4	2	Ossining	Colonial	2700	430,000

In relation to the previous home-buying example in Table 5.2, some examples of customer-specified attributes are as follows: Marital-status (categorical), Family-Size (numerical), suburban-or-city (binary), MinBedrooms (numerical), Max-Bedrooms (numerical), Max-Price (numerical) These attributes may represent either inherent customer properties (e.g., demographics), or they may specify customer requirements for the product. Such requirements are usually specified interactively during the dialog between the customer and the recommender system.

one must somehow be able to map these customer attributes/requirements into the product attributes in order to filter products for recommendation. This is achieved through the use of knowledge bases. The knowledge bases contain additional rules that map customer attributes/requirements to the product attributes:

Suburban-or-rural=Suburban $\Rightarrow$ Locality= List of relevant localities

Such rules are referred to as filter conditions because they map user requirements to the item attributes and use this mapping to filter the retrieved results.

An example of such a constraint is as follows:

Marital-status=single $\Rightarrow$ Min-Bedrooms$\leq$5

Presumably, by either domain-specific experience or through data mining of historical data sets, it has been inferred that single individuals do not prefer to buy very large houses. Similarly, a small home might not be suitable for a very large family. This constraint is modeled with the following rule:

Family-Size$\geq$5 $\Rightarrow$ Min-Bedrooms$\geq$3

there are three primary types of input to the constraint-based recommender system

1. The first class of inputs is represented by the attributes describing the inherent properties of the user (e.g., demographics, risk profiles) and specific requirements in the product (e.g., Min-Bedrooms).

The knowledge base stores information that bridges the gap between what users want and the features of available products.

Customer Attributes/Requirements: These represent user preferences and needs (e.g., location preference, minimum number of bedrooms).
Product Attributes: These are the characteristics of the items being recommended (e.g., location, number of bedrooms, price).

2. The second class of inputs is represented by knowledge bases, which map customer attributes/requirements to various product attributes. The mapping can be achieved either directly or indirectly as follows:

- Directly: These rules relate customer requirements to hard requirements on product attributes. An example of such a rule is as follows:

Suburban-or-rural=Suburban ⇒ Locality= List of relevant localities

Min-Bedrooms≥3 ⇒ Price≥100,000

If a user states a preference for a suburban location, the system can directly map that to a list of relevant localities based on pre-defined knowledge about suburban areas. Similarly, a minimum bedroom requirement translates to a minimum price threshold, considering the relationship between bedroom count and cost.

Such rules are also referred to as filter conditions.

- Indirectly: These rules relate customer attributes/requirements to typically expected product requirements. Therefore, such rules can also be viewed as an indirect way of relating customer attributes to product attributes. Examples of such rules are as follows:

Indirect mapping utilizes rules that capture typical associations between user attributes and product features. These aren't absolute requirements, but expected connections based on general knowledge.

Family-Size≥5 ⇒ Min-Bedrooms≥3 Family-Size≥5 ⇒ Min-Bathrooms≥2

A large family size (5 or more) might typically require at least 3 bedrooms and 2 bathrooms. The system leverages this knowledge to recommend products that cater to such needs, even if the user doesn't explicitly state them as requirements.

The aforementioned knowledge bases are derived from publicly available information, domain experts, past experience, or data mining of historical data sets. Therefore, a significant amount of effort is involved in building the knowledge bases

3. Finally, the product catalog contains a list of all the products together with the corresponding item attributes

You're absolutely right. In essence, constraint-based recommender systems (CBRS) can be viewed as a sophisticated data filtering process applied to the product catalog. Here's a breakdown of how it works:

1 Returning Relevant Results

From Constraints to Filters:

User-defined constraints and relevant domain rules translate into filters that narrow down the vast product catalog. Imagine each constraint or rule acting like a sieve, eliminating products that don't meet the criteria.

More simply, the set of rules and requirements can be reduced to a data filtering task on the catalog. All the customer requirements and the active rules relevant to the customer are used to construct a database selection query. The steps for creating such a filtering query are as follows

Building the Filter Query:

The system constructs a database selection query based on the combined filters. This query specifies the conditions products must meet to be included in the recommendations.

Think of it like building a complex search string that considers all the user's needs and relevant rules.

Benefits of Data Filtering Approach:

Efficiency: By leveraging filters, the system can efficiently search through a large product catalog, identifying potential matches quickly. Accuracy: The filtering process ensures only products that fulfill all the user's requirements and adhere to domain rules are considered.

Imagine a user searching for a laptop on a CBRS. They specify a minimum RAM requirement of 16GB and a price range of $1000 to $1500. Additionally, a domain rule might state that laptops under $1200 typically have less powerful processors.

Filtering Process:

Filter 1: RAM >= 16GB

Filter 2: Price >= $1000 AND Price <= $1500

Filter 3 (derived from rule): Price < $1200 => Exclude processors below a certain benchmark.

The system would then construct a database query that incorporates all these filters, effectively searching the product catalog for laptops that meet all the user's needs and considerations based on domain knowledge.

Overall, you've provided a clear and concise explanation of how CBRS leverage data filtering to identify relevant results. By translating constraints and rules into filters, the system can efficiently search through a vast product catalog and deliver accurate recommendations that strictly adhere to user requirements.

antecedent - preceding condition

1. For each requirement (or personal attribute) specified by the customer in their user interface, it is checked whether it matches the antecedent of a rule in the knowledge base. If such a matching exists, then the consequent of that rule is treated as a valid selection condition. For example, consider the aforementioned real-estate example. If the customer has specified Family-Size=6 and ZIP Code=10547 among their personal attributes and preferences in the user interface, then it is detected that

Family-Size=6

triggers the following rules:

Family-Size≥5 ⇒ Min-Bedrooms≥3 Family-Size≥5 ⇒ Min-Bathrooms≥2

Therefore, the consequents of these conditions are added to the user requirements. The rule base is again checked with these expanded requirements, and it is noticed that the newly added constraint Min-Bedrooms≥ 3 triggers the following rules:

Min-Bedrooms≥3 ⇒ Price≥100,000 Min-Bedrooms≥3 ⇒ Bedrooms≥3 Min-Bathrooms≥3 ⇒ Bathrooms≥2 Therefore, the conditions Price≥100,000,

and the range constraints on the requirement attributes Min-Bedrooms and Min-Bathrooms are replaced with those on the product attributes Bedrooms and Bathrooms. In the next iteration, it is found that no further conditions can be added to the user requirements.

2. These expanded requirements are used to construct a database query in conjunctive normal form. This represents a traditional database selection query, which computes the intersection of the following constraints on the product catalog:

$$(\text{Bedrooms} \geq 3) \wedge (\text{Bathrooms} \geq 2) \wedge (\text{Price} \geq 100{,}000) \wedge (\text{ZIP Code} = 10547)$$

Note that the approach essentially maps all customer attribute constraints and requirement attribute constraints to constraints in the product domain.

3. This selection query is then used to retrieve the instances in the catalog that are relevant to the user requirements

2. Interaction Approach

1. An interactive interface is used by the user to specify her initial preferences. A common approach is to use a Web style form in which the desired values of the attributes may be entered.

2. The user is presented with a ranked list of matching items. An explanation for why the items are returned is typically provided. In some cases, no items might match the user requirements. In such cases, possible relaxations of the requirements might be suggested

3. The user then refines her requirements depending on the returned results. This refinement might take the form of the addition of further requirements, or the removal of some of the requirements. For example, when an empty set is returned, it is evident that some of the requirements need to be relaxed. Constraint satisfaction methods are used to identify possible sets of candidate constraints, which might need to be relaxed. Therefore, the system generally helps the user in making her modifications in a more intelligent and efficient way.

3. Ranking the Matched Items

Using a single attribute has the drawback that the importance of other attributes is discounted. A common approach is to use utility functions in order to rank the matched items. Let $\overline{V} = (v_1 \ldots v_d)$ be the vector of values defining the attributes of the matched products. Therefore, the dimensionality of the content space is d. The utility functions may be defined as weighted functions of the utilities of individual attributes. Each attribute has a weight w_j assigned to it, and it has a contribution defined by the function $f_j(v_j)$ depending on the value v_j of the matched attribute. Then, the utility $U(\overline{V})$ of the matched item is given by the following:

$$U(\overline{V}) = \sum_{j=1}^{d} w_j \cdot f_j(v_j) \tag{5.1}$$

Clearly, one needs to instantiate the values of w_j and $f_j(\cdot)$ in order to learn the utility function. The design of effective utility functions often requires domain-specific knowledge, or learning data from past user interactions. For example, when v_j is numeric, one might assume that the function $f_j(v_j)$ is linear in v_j, and then learn the coefficients of the linear

Handling Unacceptable Results or Empty Sets

Imagine a user searching for a laptop under $1000 with at least 16GB of RAM. The initial filters might yield no results. The system could:
Suggest increasing the budget slightly or consider laptops with 8GB of RAM (if expandable) while highlighting the trade-offs.
Another example: A user searches for dresses in a specific color and size. If options are limited, the system could:
Recommend similar styles or dresses in different colors that might still be of interest.

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.

Most of these methods use similar principles; small sets of violating constraints are determined, and the most appropriate relaxations are suggested based on some pre-defined criteria. In real applications, however, it is sometimes difficult to suggest concrete criteria for constraint relaxation. Therefore, a simple alternative is to present the user with small sets of inconsistent constraints, which can often provide sufficient intuition to the user in formulating modified constraints

Adding Constraints

In some cases, the number of returned results may be very large, and the user may need to suggest possible constraints to be added to the query. In such cases, a variety of methods can be used to suggest constraints to the user along with possible default values. The attributes for such constraints are often chosen by mining historical session logs. The historical session logs can either be defined over all users, or over the particular user at hand. The latter provides more personalized results, but may often be unavailable for infrequently bought items (e.g., cars or houses). It is noteworthy that knowledge-based systems are generally designed to not use such persistent and historical information precisely because they are designed to work in cold-start settings; nevertheless, such information can often be very useful in improving the user experience when it is available.

Case-Based Recommenders

In case-based recommenders, similarity metrics are used to retrieve examples that are similar to the specified targets (or cases).

The user might specify a locality, the number of bedrooms, and a desired price to specify a target set of attributes. Unlike constraint-based systems, no hard constraints (e.g., minimum or maximum values) are enforced on these attributes.

It is also possible to design an initial query interface in which examples of relevant items are used as targets. However, it is more natural to specify desired properties in the initial query interface.

A similarity function is used to retrieve the examples that are most similar to the user-specified target. For example, if no homes are found specifying the user requirements exactly, then the similarity function is used to retrieve and rank items that are as similar as possible to the user query. Therefore, unlike constraint-based recommenders, the problem of retrieving empty sets is not an issue in case-based recommenders.

In order for a case-based recommender system to work effectively, there are two crucial aspects of the system that must be designed effectively:

1. Similarity metrics: The effective design of similarity metrics is very important in casebased systems in order to retrieve relevant results. The importance of various attributes must be properly incorporated within the similarity function for the system to work effectively.

2. Critiquing methods: The interactive exploration of the item space is supported with the use of critiquing methods. A variety of different critiquing methods are available to support different exploration goals

Similarity Metrics

by domain experts, or can be tweaked by a learning process.

Consider an application in which the product is described by d attributes. We would like to determine the similarity values between two *partial* attribute vectors defined on a subset S of the universe of d attributes (i.e., $|S| = s \leq d$). Let $\overline{X} = (x_1 \ldots x_d)$ and $\overline{T} = (t_1 \ldots t_d)$ represent two d-dimensional vectors, which might be partially specified. Here, $\overline{T}$ represents the target. It is assumed that at least the attribute subset $S \subseteq \{1 \ldots d\}$ is specified in both vectors. Note that we are using *partial* attribute vectors because such queries are often defined only on a small subset of attributes specified by the user. For example, in the aforementioned real estate example, the user might specify only a small set of query features, such as the number of bedrooms or bathrooms. Then, the similarity function $f(\overline{T}, \overline{X})$ between the two sets of vectors is defined as follows:

$$f(\overline{T}, \overline{X}) = \frac{\sum_{i \in S} w_i \cdot Sim(t_i, x_i)}{\sum_{i \in S} w_i}$$

how similar two attributes are

Fist, we will discuss the determination of the similarity function Sim(ti, xi). Note that these attributes might be either quantitative or categorical, which further adds to the heterogeneity and complexity of such a system. Furthermore, attributes might be symmetric or asymmetric in terms of higher or lower values

Other attributes might be completely symmetric, in which case the user would want the attribute value exactly at the target value ti. An example of a symmetric metric is as follows:

$$Sim(t_i, x_i) = 1 - \frac{|t_i - x_i|}{max_i - min_i}$$

Here, maxi and mini represent the maximum or minimum possible values of the attribute i. Alternatively, one might use the standard deviation σi (on historical data) to set the similarity function:

$$Sim(t_i, x_i) = \max\left\{0, 1 - \frac{|t_i - x_i|}{3 \cdot \sigma_i}\right\}$$

Note that in the case of the symmetric metric, the similarity is entirely defined by the difference between the two attributes. In the case of an asymmetric attribute, one can add an additional asymmetric reward, which kicks in depending on whether the target attribute value is smaller or larger. For the case of attributes in which larger values are better, an example of a possible similarity function is as follows:

$$Sim(t_i, x_i) = 1 - \frac{|t_i - x_i|}{max_i - min_i} + \underbrace{\alpha_i \cdot I(x_i < t_i) \cdot \frac{|t_i - x_i|}{max_i - min_i}}_{\text{Asymmetric reward}}$$

Critiquing Methods

Critiques are motivated by the fact that users are often not in a position to state their requirements exactly in the initial query. In some complex domains, they might even find it difficult to translate their needs in a semantically meaningful way to the attribute values in the product domain. It is only after viewing the results of a query that a user might realize that she should have couched her query somewhat differently. Critiques are designed to provide the users this ability after the fact.

To provide the users this ability after the fact. After the results have been presented to the users, feedback is typically enabled through the use of critiques. In many cases, the interfaces are designed to critique the most similar matching item, although it is technically possible for the user to critique any of the items on the retrieved list of k items. In critiques, the users specify change requests on one or more attributes of an item that they may like. For example, in the home-buying application of Figure 5.2, the user might like a particular house, but she may want the house in a different locality or with one more bedroom. Therefore, the user may specify the changes in the features of one of the items she likes. The user may specify a directional critique (e.g., "cheaper") or a replacement critique (e.g., "different color").

At a given moment in time, the user may specify either a single feature or a combination of features for modification. In this context, the critiques are of three different types, corresponding to

- simple critiques,
- compound critiques, and
- dynamic critiques.

Simple Critiques

In a simple critique, the user specifies a single change to one of the features of a recommended item.

Compound Critiques: Making Car Navigation Faster and More Efficient
Imagine you're shopping for a new car online. It can be overwhelming with so many options, right? Here's how compound critiques can streamline your search:
Problem: Traditional systems often require multiple steps to refine your search. You might choose a price range, then adjust features separately (e.g., size, interior).
Solution: Compound critiques allow you to specify multiple changes in one go.
Example: The "Car Navigator" system understands these combined preferences. You can say "classier, roomier, cheaper" – all at once!

Compound Critiques

Compound critiques were developed to reduce the length of recommendation cycles. In this case, the user is able to specify multiple feature modifications in a single cycle. For example, the Car Navigator system allows the user to specify multiple modifications, which are hidden behind informal descriptions that the user can understand (e.g., classier, roomier, cheaper, sportier). For example, the domain expert might encode the fact that “classier” suggests a certain subset of models with increased price and sophisticated interior structure.

Hidden Meanings: Behind the scenes, the system decodes these informal terms. "Classier" might translate to specific car models with higher prices and luxurious interiors.
Benefits:
Faster Search: You express your desires more efficiently, reducing the number of steps needed to find the perfect car.
More Natural Interaction: The system understands everyday language, making the search feel more intuitive.
Better Results: By considering all your preferences together, the system can recommend cars that truly match your vision.
Overall, compound critiques offer a more user-friendly and efficient way to navigate car options. You can express your ideal car in a natural way, and the system works behind the scenes to find the perfect match.

Dynamic Critiques

Although compound critiques allow larger jumps through the navigation space, they do have the drawback that the critiquing options presented to the user are static in the sense that they do not depend on the retrieved results. For example, if the user is browsing cars, and she is already browsing the most expensive car with the largest horsepower possible, the option to increase the horsepower and the price will still be shown in the critiquing interface. Clearly, specifying these options will lead to a fruitless search. This is because users are often not fully aware of the inherent trade-offs in the complex product space. In dynamic critiquing, the goal is to use data mining on the retrieved results to determine the most fruitful avenues of exploration and present them to the user. Thus, dynamic critiques are, by definition, compound critiques because they almost always represent combinations of changes presented to the user. The main difference is that only the subset of the most relevant possibilities are presented, based on the currently retrieved results. Therefore, dynamic critiques are designed to provide better guidance to the user during the search process.

Extra slides

Dynamic Critiques: Guiding Your Car Search Even Smarter
Imagine you're still searching for the perfect car using the "Car Navigator" system. Compound critiques were a good start, but there's room for improvement:

The Challenge of Compound Critiques:

While compound critiques allow you to combine preferences, they offer pre-defined options regardless of the current search results.
For example, if you see the most expensive, most powerful car already, the system might still suggest increasing those very features – a dead end.
Enter Dynamic Critiques:

These critiques use data analysis to understand the retrieved car options.
Based on the current results, the system suggests the most relevant combination of changes to explore.
Think of it as the system learning from the available cars and guiding your search more effectively.
Dynamic vs. Compound Critiques:

Both are compound critiques (multiple changes in one go).
The key difference: dynamic critiques are adaptive. They analyze the current search results to suggest the most relevant combinations for further exploration.
Benefits of Dynamic Critiques:

Smarter Navigation: Avoids suggesting pointless changes like increasing price and horsepower for the already top-of-the-line cars.
Better Guidance: The system understands the trade-offs within the car options, suggesting more fruitful combinations to explore your needs.
Overall, dynamic critiques offer a more intelligent and user-friendly search experience. The system personalizes suggestions based on the available cars, helping you find the perfect match faster.

Recommendation as Classification

You're absolutely right. Recommendation systems can be approached from both a classification and a case-based perspective. Here's a breakdown of how recommendation systems can be viewed as classification problems:

The Goal: Predict a user's preference (like/dislike) for an item.

The Approach:

- classification problem: features→like/dislike (rating)
- use of general machine learning techniques
  - probabilistic methods – Naive Bayes linear
  - classifiers
  - decision trees neural
  - networks
  - . . .

Features:
We represent items and users as a set of features. These features can include:
Item features: Product description, category, brand, price, etc.
User features: Past purchase history, browsing behavior, demographics, etc.
Machine Learning Techniques:
We train a model using historical data where items are labeled as liked/disliked by users. This data serves as training examples for the model.
Common machine learning techniques used for classification include:
Probabilistic methods: Naive Bayes (calculates the probability of an item being liked based on its features)
Linear classifiers: Logistic Regression (classifies items as liked/disliked based on a linear combination of features)
Decision Trees: (classifies items by asking a series of questions about their features)
Neural Networks: Powerful models that can learn complex relationships between features and preferences.

wider context: machine learning techniques

Recommendation as Classification

Benefits:

Scalability: These techniques can handle large datasets of users and items.
Adaptability: The model can learn and adapt to changing user preferences over time.
Personalization: By considering user features, recommendations can be tailored to individual preferences.
Wider Context:

- classification problem: features→like/dislike (rating)
- use of general machine learning techniques
  - probabilistic methods – Naive Bayes linear
  - classifiers
  - decision trees neural
  - networks
  - . . .

wider context: machine learning techniques

While classification is a powerful approach, recommendation systems often incorporate other techniques like collaborative filtering (finding similar users or items) or content-based filtering (recommending items similar to what the user liked in the past) to improve results.

Overall, viewing recommendations as classification problems allows leveraging powerful machine learning techniques to predict user preferences and personalize recommendations.

Content-Based Recommendations: Advantages

- user independence – does not depend on other users
- new items can be easily incorporated (no cold start)
- transparency – understandable, provides explanations (at least with basic methods)

Content-Based Recommendations: Limitations

Limited Content Analysis:
Content-based systems rely on analyzing the content associated with items. This can be effective for textual data like product descriptions or reviews, but it struggles with:
Multimedia: Images, videos, and audio require specialized techniques for content extraction and understanding.
Complex Content: Content analysis might not capture the full essence of an item, especially for creative or subjective domains like art or music.

Overspecialization - "Echo Chamber" Effect:

Content-based systems can become too good at recommending similar items based on past preferences. This can lead to an "echo chamber" effect, where users are only exposed to items that reinforce their existing interests, limiting their ability to discover new things.

- limited content analysis
  - content may not be automatically extractable (multimedia)
  - missing domain knowledge
  - keywords may not be sufficient

Missing Domain Knowledge:
These systems rely heavily on features extracted from the content. However, they might lack the deeper understanding of a human expert.
For instance, keywords alone might not capture the true value of an item. A simple recipe recommendation system based on keywords might miss a gourmet dish with unusual ingredients but clear instructions.

- overspecialization – "more of the same", too similar items
- new user – ratings or information about user has to be collected

Challenges with New Users:

These systems typically require some user data (ratings, browsing history) to build a user profile and make recommendations.
This creates a cold start problem for new users where there's not enough information to make personalized recommendations.

Content-Based vs Collaborative Filtering

- paper "Recommending new movies: even a few ratings are more valuable than metadata"
- (context: Netflix)

  our experience in educational domain – difficulty rating (Sokoban, countries)

Knowledge-based Recommendations

application domains:

- expensive items, not frequently purchased, few ratings (car, house)
- time span important (technological products) explicit requirements of user (vacation)

- collaborative filtering unusable – not enought data content based – “similarity” not sufficient

Knowledge-based Recommendations

- constraint-based
  - explicitly defined conditions
- case-based
  - similarity to specified requirements

"conversational" recommendations

Constraint-Based Recommendations – Example

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

Constraint Satisfaction Problem

- V is a set of variables
- D is a set of finite domains of these variables
- C is a set of constraints

Typical problems: logic puzzles (Sudoku, N-queen), scheduling

CSP: N-queens

problem: place N queens on an $N \times N$ chess-board, no two queens threaten each other

- V – N variables (locations of queens)
- D – each domain is $\{1, \ldots, N\}$
- C – threatening

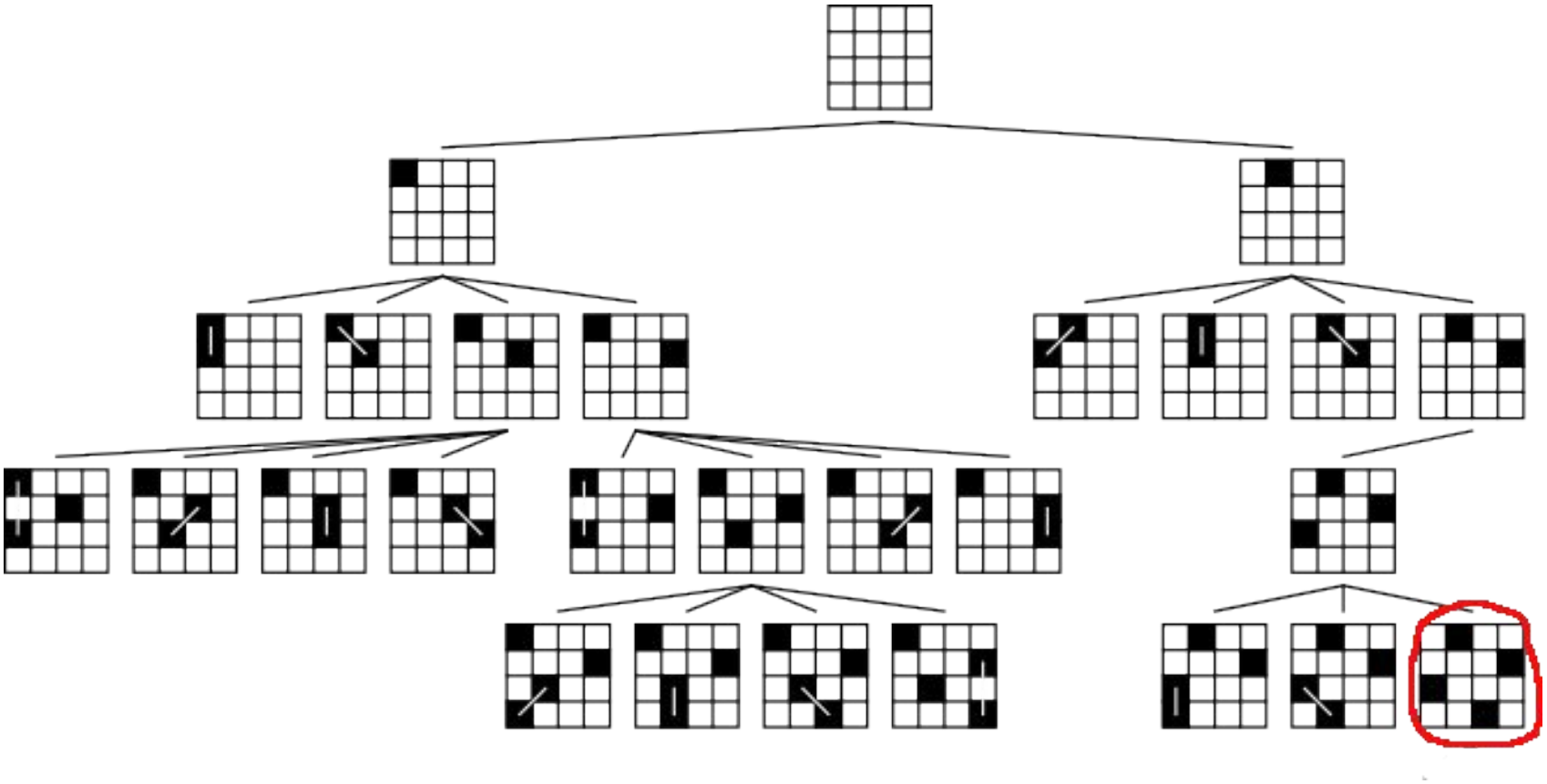
The domain for each variable is simply {1, 2, ..., N}, representing the N rows on the chessboard where a queen can be placed.

Constraints (C): The constraints are the key element that captures the "no two queens threaten each other" rule. We can define two types of constraints:
No row conflicts: Two queens cannot be in the same row. This can be enforced by ensuring no two variables have the same value (i.e., no two queens are placed in the same row).
No diagonal conflicts: No two queens can be positioned diagonally such that they can attack each other. This is a bit trickier to define as a constraint, but we can represent it mathematically. A queen at position (row i, column j) can attack another queen at (row k, column l) if the absolute difference between their row positions (i - k) is equal to the absolute difference between their column positions (j - l).

CSP Algorithms

- basic algorithm –
- backtracking heuristics
  - preference for some
  - branches pruning
  - ... many others

CSP Example: N-queens Problem



Recommender Knowledge Base

represent different components of a knowledge base used in recommender systems, particularly constraint-based recommender systems.

Customer Properties (VC):
This refers to the characteristics or attributes associated with a customer. These can be demographic information (age, location), purchase history, preferences indicated through ratings or browsing behavior, and loyalty program data.
Product Properties (VPROD):
This represents the features or attributes of a product or item being recommended. Examples include price, brand, category, technical specifications, size, color, and any other relevant product characteristic.
Constraints (CR):
These are limitations or preferences set by the customer on the recommended products. They can be based on customer properties. For instance, a constraint might be "price less than $100" or "only consider products with a 5-star rating."

- customer properties V_C
- product properties V_{PROD}
- constraints C_R (on customer properties)
- filter conditions C_F – relationship between customer and product
- products C_{PROD} – possible instantiations

Filter Conditions (CF):
These are additional criteria used to narrow down the search space before applying constraints. They might be based on product properties. For instance, a filter condition could be "only show laptops" or "exclude out-of-stock items."
Customer-Product Relationship (CPROD):
This represents the connection between customers and products. It might be explicit (e.g., a purchase history record) or implicit (e.g., browsing behavior indicating interest).
Possible Product Instances (CPROD):
This refers to the actual set of products or items that the system can recommend based on the available customer and product data. It's the pool of products from which recommendations are drawn after applying constraints and filters.

V_C = {kl_c: [expert, average, beginner] ..... /* level of expertise */
wr_c: [low, medium, high] ..... /* willingness to take risks */
id_c: [shortterm, mediumterm, longterm] ..... /* duration of investment */
aw_c: [yes, no] ..... /* advisory wanted ? */
ds_c: [savings, bonds, stockfunds, singleshares] ..... /* direct product search */
sl_c: [savings, bonds] ..... /* type of low-risk investment */
av_c: [yes, no] ..... /* availability of funds */
sh_c: [stockfunds, singlshares] ..... /* type of high-risk investment */ }

V_{PROD} = {$name_p$: [text] ..... /* name of the product */
er_p: [1..40] ..... /* expected return rate */
ri_p: [low, medium, high] ..... /* risk level */
$mniv_p$: [1..14] ..... /* minimum investment period of product in years */
$inst_p$: [text] ..... /* financial institute */ }

Recommender Systems Handbook; Developing Constraint-based Recommenders

$$C_R = \{CR_1: wr_c = high \rightarrow id_c \neq shortterm,$$
$$CR_2: kl_c = beginner \rightarrow wr_c \neq high\}$$

$$C_F = \{CF_1: id_c = shortterm \rightarrow mniv_p < 3,$$
$$CF_2: id_c = mediumterm \rightarrow mniv_p \geq 3 \wedge mniv_p < 6,$$
$$CF_3: id_c = longterm \rightarrow mniv_p \geq 6,$$
$$CF_4: wr_c = low \rightarrow ri_p = low,$$
$$CF_5: wr_c = medium \rightarrow ri_p = low \vee ri_p = medium,$$
$$CF_6: wr_c = high \rightarrow ri_p = low \vee ri_p = medium \vee ri_p = high,$$
$$CF_7: kl_c = beginner \rightarrow ri_p \neq high,$$
$$CF_8: sl_c = savings \rightarrow name_p = savings,$$
$$CF_9: sl_c = bonds \rightarrow name_p = bonds\ \}$$

$$C_{PROD} = \{CPROD_1: name_p = savings \wedge er_p = 3 \wedge ri_p = low \wedge mniv_p = 1 \wedge inst_p = A;$$
$$CPROD_2: name_p = bonds \wedge er_p = 5 \wedge ri_p = medium \wedge mniv_p = 5 \wedge inst_p = B;$$
$$CPROD_3: name_p = equity \wedge er_p = 9 \wedge ri_p = high \wedge mniv_p = 10 \wedge inst_p = B\}$$

Recommender Systems Handbook; Developing Constraint-based Recommenders

Development of Knowledge Bases

Rapid Prototyping:
Build a basic knowledge base with minimal functionality initially.
Get feedback from users and domain experts to refine and iterate on the knowledge base.
This helps identify and address issues early in the development process.
Incremental Development:
Break down the knowledge base development into smaller, manageable tasks.
Focus on building core functionalities first, then gradually add complexity.
Knowledge Acquisition Techniques:
Utilize structured interviews with domain experts.

- difficult, expensive
- specilized graphical
- tools

methodology (rapid prototyping, detection of faulty constraints, ...)

Leverage existing data sources like customer records and product manuals.
Employ machine learning techniques to extract knowledge from text (e.g., product descriptions).
Data Validation and Cleaning:
Implement data quality checks to identify and correct errors or inconsistencies.
Establish data cleansing procedures to ensure high-quality data in the knowledge base.
Specialized Graphical Tools:
Utilize knowledge base development tools that offer visual interfaces for easier knowledge representation and editing.
These tools can help domain experts who might not have programming expertise contribute to the knowledge base creation.
Constraint Detection and Debugging:
Integrate mechanisms to identify faulty constraints or knowledge inconsistencies.
Provide functionalities for diagnosing and resolving issues within the knowledge base.

Unsatisfied Requirements

no solution to provided constraints

- we want to provide user at least
- something constraint relaxation
- proposing “repairs”
- minimal set of requirements to be changed

User Guidance

There are various approaches to user guidance, each with its own advantages and disadvantages when it comes to requirements elicitation for a recommender system. Here's a breakdown of the methods you mentioned:
Requirements Elicitation Process:
This is the initial stage where you gather information about user needs and preferences to understand what kind of guidance they require for interacting with the recommender system.

For simple systems or basic user profiles, static forms or session-independent profiles might suffice.
For more complex systems or when contextual guidance is important, consider conversational dialogues.

requirements elicitation process

- session independent user profile static fill-out forms
- conversational dialogs

Conversational Dialogs:

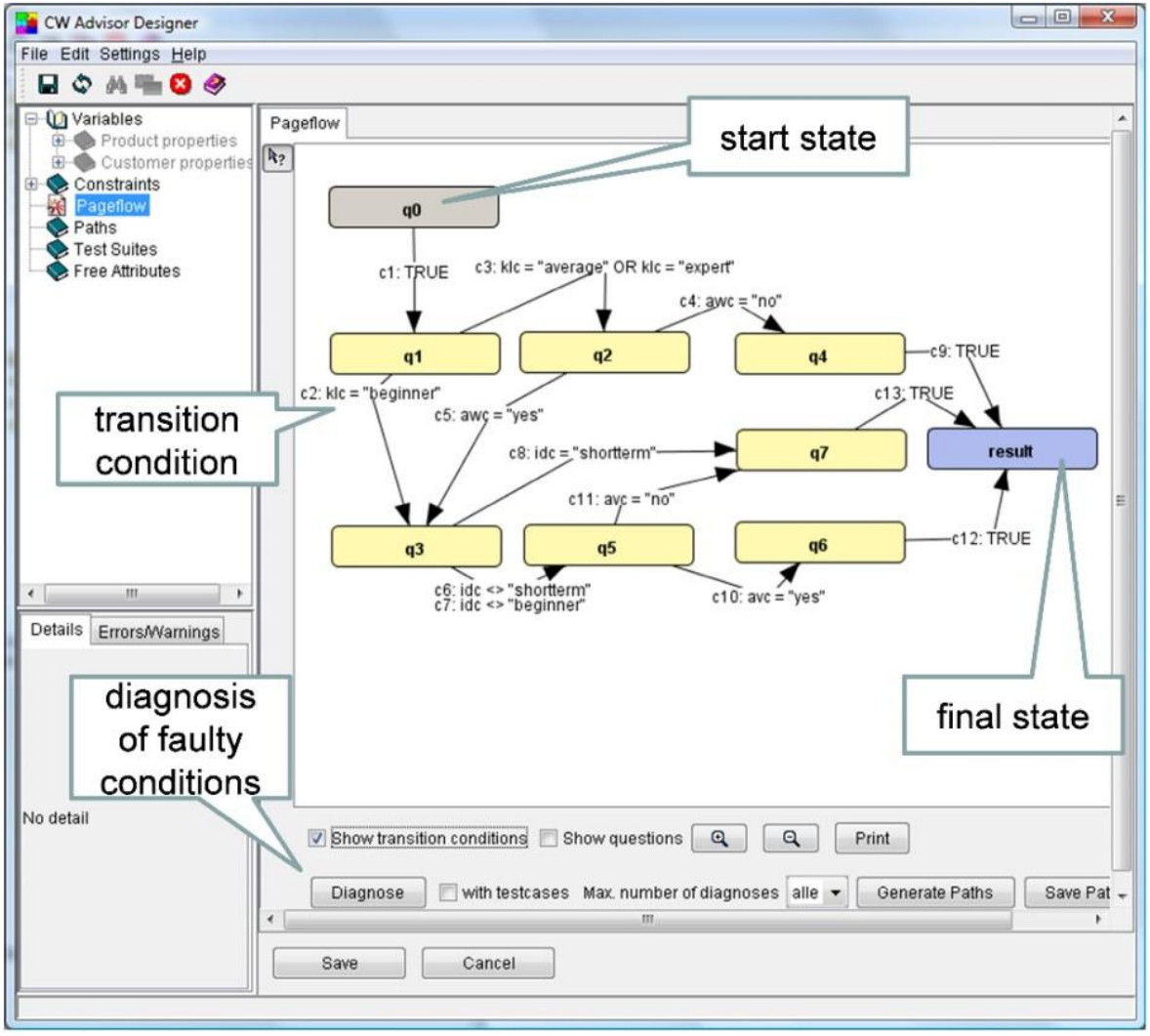
Method: This approach uses interactive dialogues throughout the user's interaction with the system. The system asks clarifying questions to understand user needs and preferences in real-time.
Advantages:
More engaging and dynamic user experience.
Can gather context-specific information based on the user's current search or browsing behavior.
Disadvantages:
Requires more sophisticated natural language processing (NLP) capabilities in the system.
Might feel intrusive if not implemented carefully.

Session-Independent User Profile:
Method: This approach gathers user information through static forms or surveys that users fill out independently, outside of their interaction with the system.
Advantages:
Easy to implement and manage.
Can collect a wide range of demographic and preference data upfront.
Disadvantages:
Limited ability to capture user context or specific needs during an active search session.
Users might provide inaccurate or incomplete information if the forms are lengthy or impersonal.

Static Fill-Out Forms:

Method: This is a specific type of session-independent approach where users interact with static forms within the system to provide their preferences.
Advantages:
Can be integrated directly into the user interface.
Offers some control over the data collected.
Disadvantages:
Similar limitations to session-independent profiles – lack of contextual information and potential for user fatigue with long forms.

User Guidance



Recommender Systems Handbook; Developing Constraint-based Recommenders

User Guidance

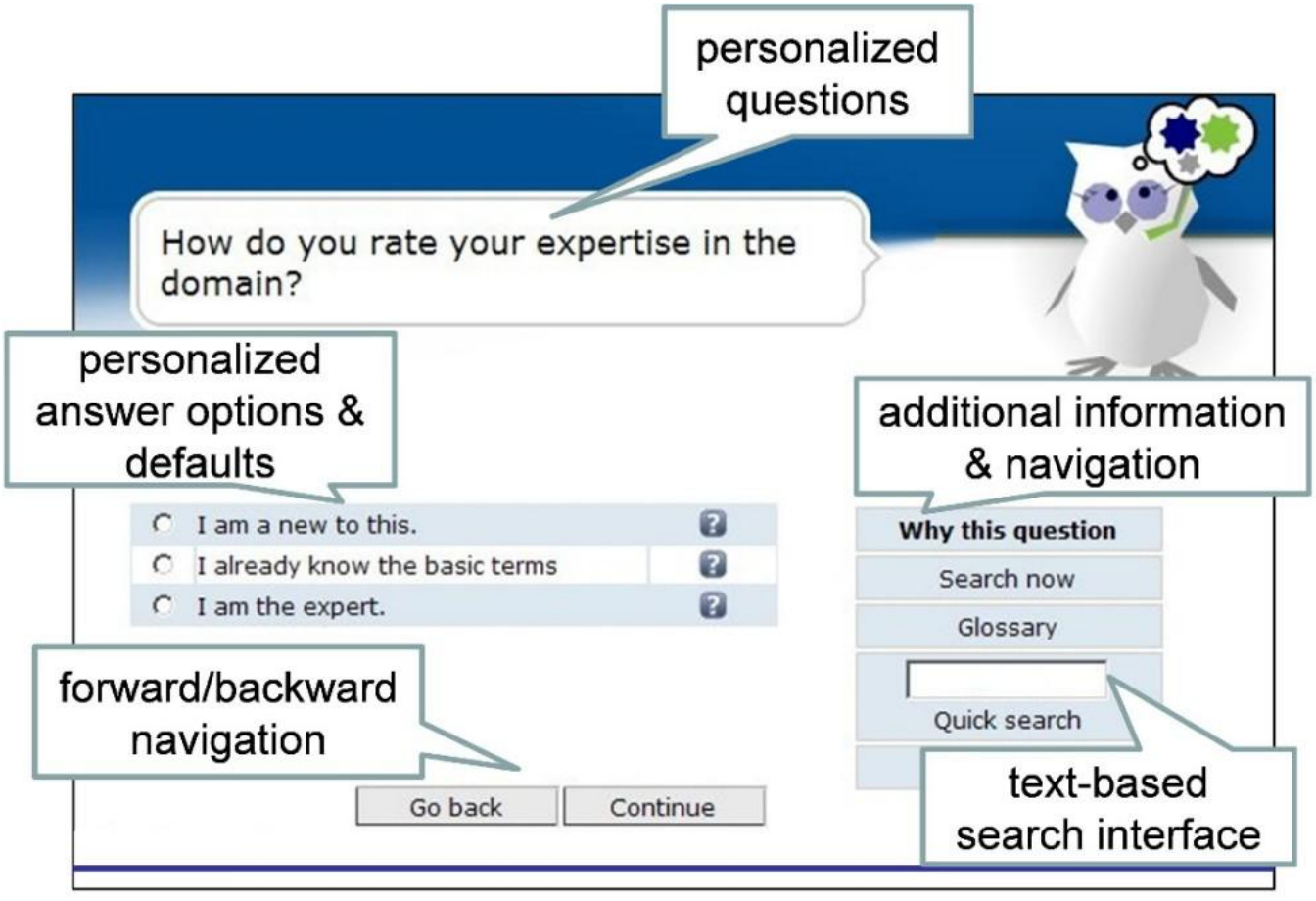
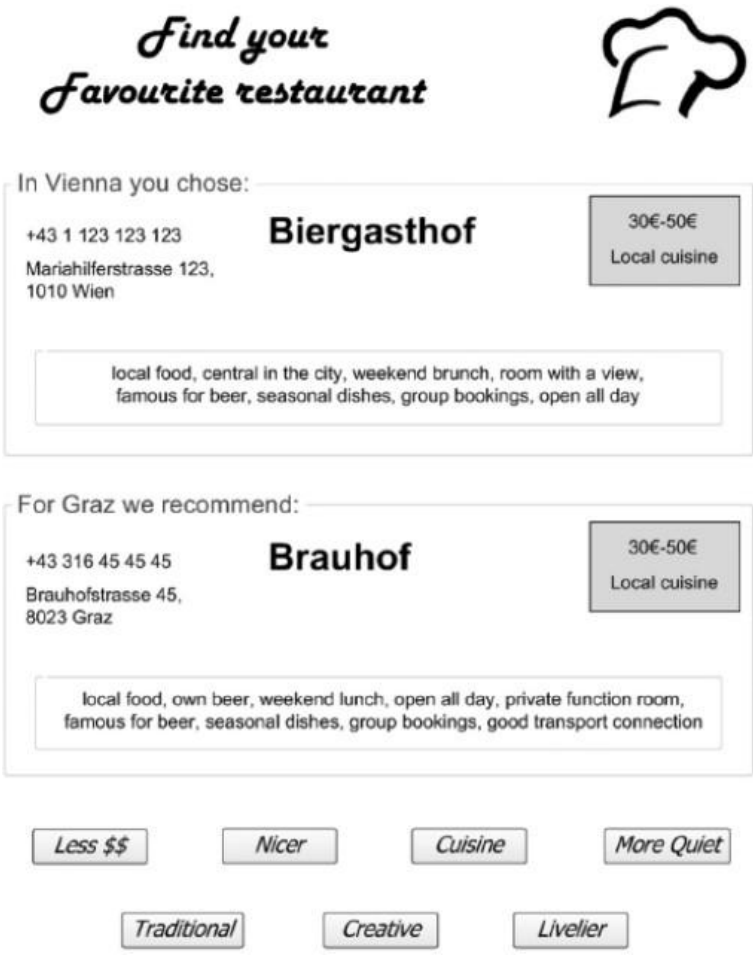


Fig. 6.4: Interactive and personalized preference elicitation example. Customers specify their preferences by answering questions.

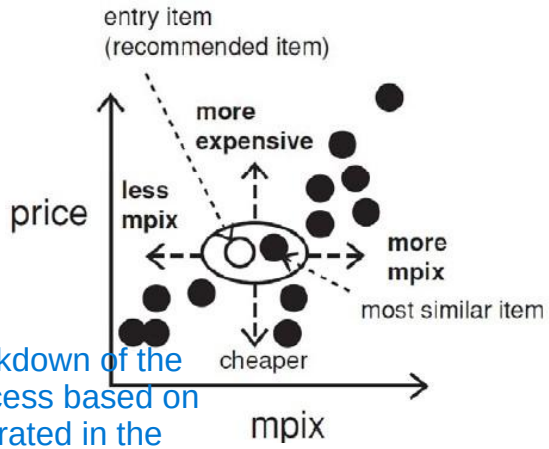
Recommender Systems Handbook; Developing Constraint-based Recommenders

Critiquing

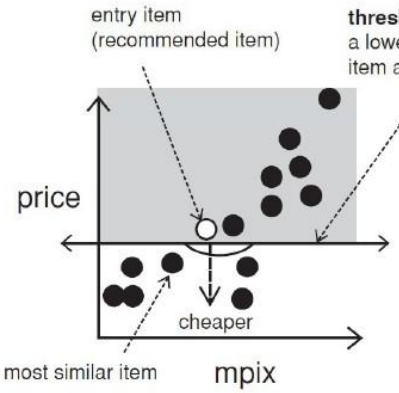

Find your
Favourite restaurant
In Vienna you chose:
+43 1 123 123 123
Mariahilferstrasse 123,
1010 Wien
Biergasthof
30€-50€
Local cuisine
local food, central in the city, weekend brunch, room with a view,
famous for beer, seasonal dishes, group bookings, open all day
For Graz we recommend:
+43 316 45 45 45
Brauhofstrasse 45,
8023 Graz
Brauhof
30€-50€
Local cuisine
local food, own beer, weekend lunch, open all day, private function room,
famous for beer, seasonal dishes, group bookings, good transport connection
Less $$
Nicer
Cuisine
More Quiet
Traditional
Creative
Livelier

Critiquing

The image you sent depicts a concept in recommender systems called critiquing. It refers to a method that allows users to refine or improve recommendations made by a recommender system. The image shows a critiquing process based on price.



Critique on price

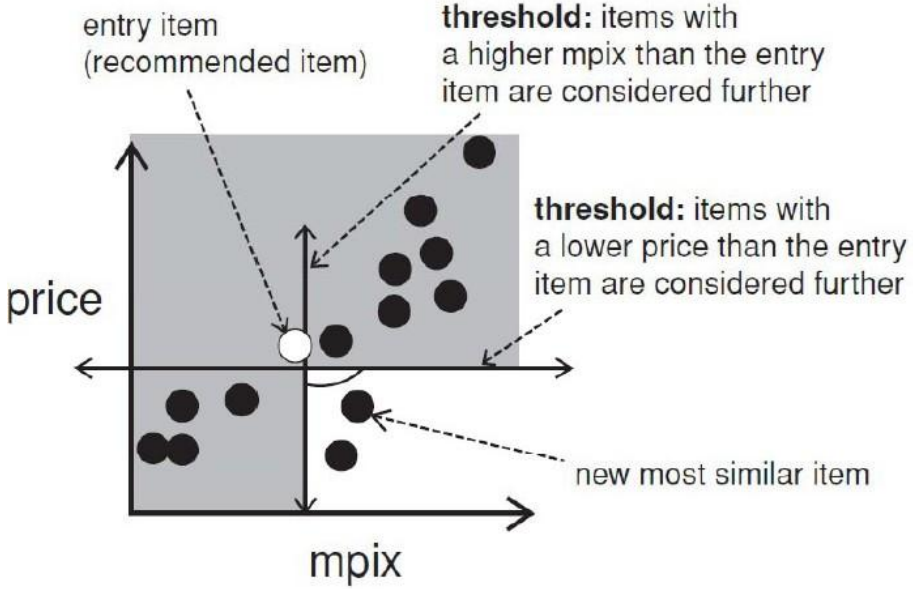


Here's a breakdown of the critiquing process based on price, as illustrated in the image:

The system initially recommends an item (entry item), which is considered the most similar item based on an unspecified measure of similarity.

The user is given the option to critique the recommendation based on price. They can indicate if they prefer cheaper or more expensive options.

Based on the user's critique, the system refines the recommendation. If the user prefers cheaper options, the system finds an item with a lower price than the entry item (but most similar to it) and presents it as the new most similar item.



The converse is also true: if the user indicates a preference for more expensive options, the system searches for an item with a higher price (but most similar to the entry item) and presents it as the new most similar item.

This process can be repeated until the user finds a satisfactory recommendation.

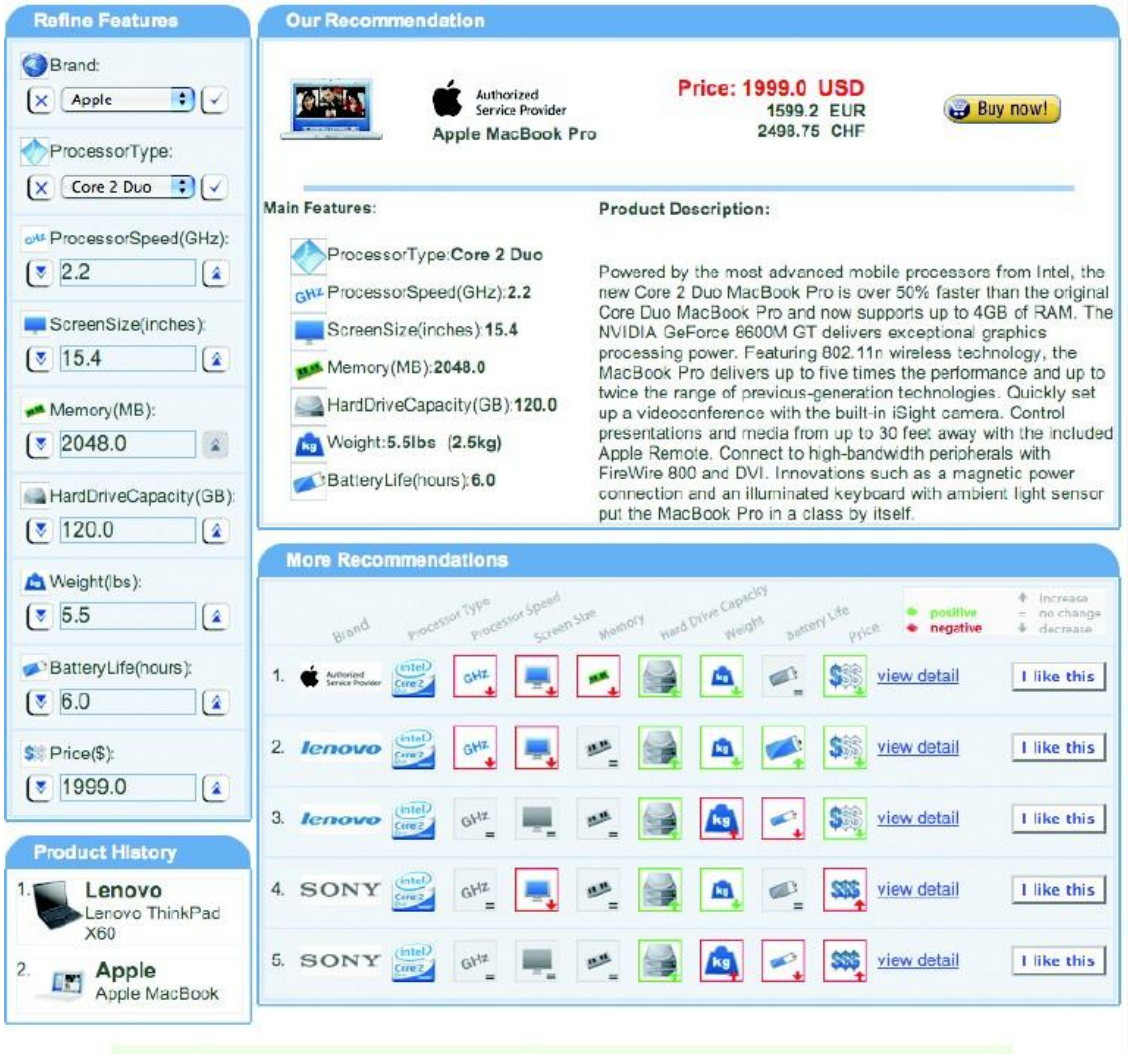
Critiquing systems can be helpful for users who have a specific preference but may not be able to articulate exactly what they are looking for. They can be especially useful for recommender systems that deal with a vast array of items, where price can be a major differentiating factor.

Critiquing: Example

The image you sent is a screenshot of a webpage that shows an example of a critiquing-based recommender system for laptops. The recommender system suggests a laptop (Apple MacBook Pro) and the user can provide feedback (critiques) to refine the recommendation based on their preferences.

In this example, the critiquing feature focuses on four attributes: Brand, Processor Speed (GHz), Memory (MB), and Hard Drive Capacity (GB). Here's a breakdown of what's displayed on the screen:

Rafino Features: This section displays the attributes (Brand, Processor Speed, Memory, Hard Drive Capacity) of the recommended laptop (Apple MacBook Pro). It also shows the corresponding values for each attribute. For instance, the Processor Speed is 2.2 GHz.
Our Recommendation: This section displays the recommended laptop, which is the Apple MacBook Pro in this case. It also shows its price (USD 1999.00) and a button labeled "Buy Now!".



More Recommendations: This section lists three other laptops (Jenovo, lenovo, and SONY) that the user can explore by clicking "view detail".
Product History: This section shows previously viewed laptops (Apple MacBook and Lenovo ThinkPad X60) by the user.
Critiquing: This section allows the user to provide feedback (critiques) on the recommended laptop (Apple MacBook Pro) based on the four attributes mentioned earlier. Unfortunately, the image doesn't show how the user would critique the recommendation.

Critiquing: Example

Critiquing recommender systems can be helpful for users who are unsure exactly what they're looking for in a laptop but may have preferences on certain features. By critiquing the system's recommendations, users can iteratively narrow down their search options until they find a laptop that meets their needs.

The image you sent is a screenshot of a comparison window used to compare a house with other apartments. The window allows users to compare the house with other apartments for a variety of factors, including size, location, and price. Let's break down the different sections of the window:
Compare : This is the title at the top of the window.
Would you like to compare Apt 34 : This section displays the details of the house being considered, which is Apt 34. It includes the type of apartment (room in a house), size (15 square meters), amenities (private bathroom and private kitchen), distance to the workplace (15 minutes), and price (600 frs).

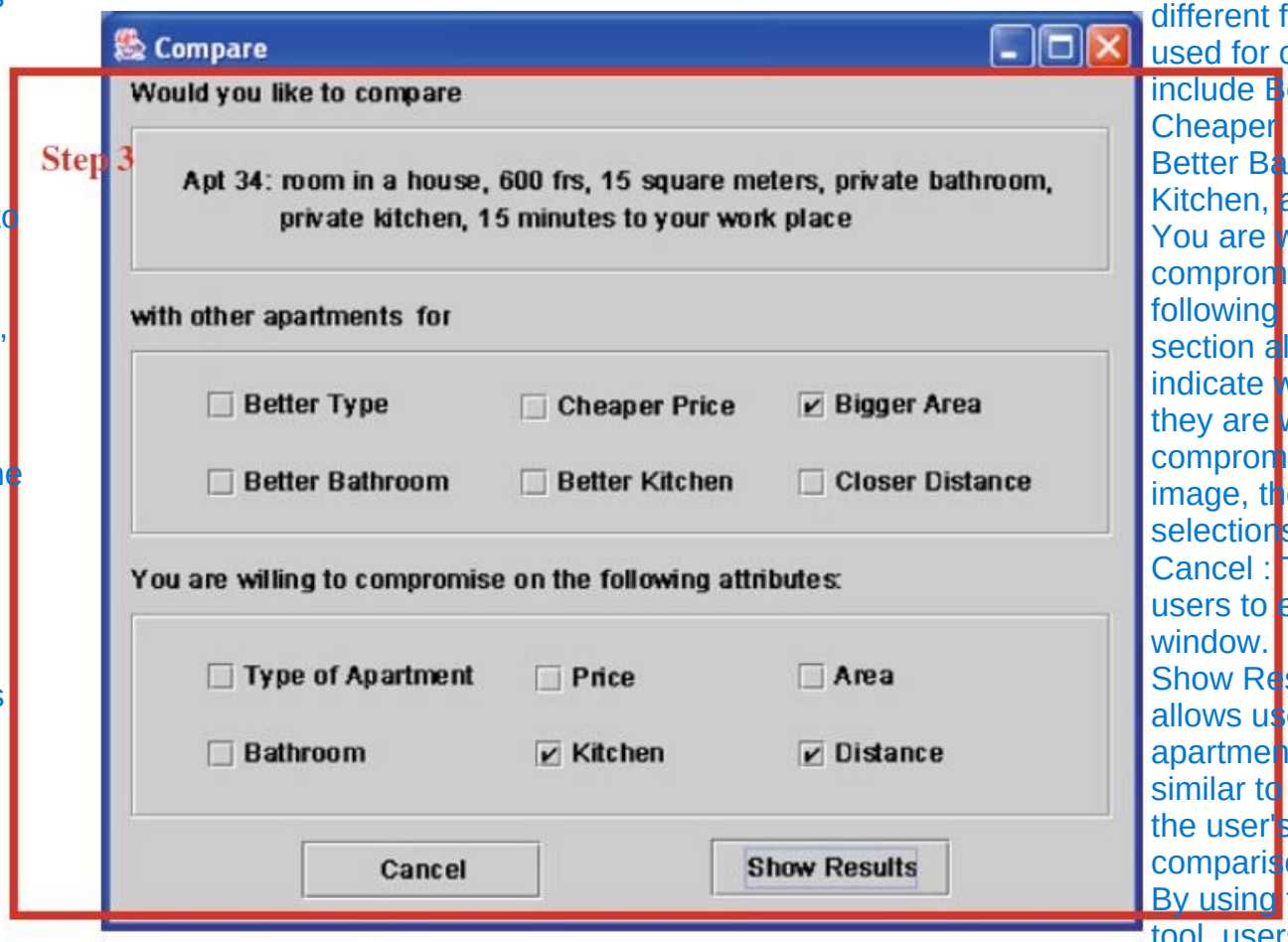


Fig. 3 Critiquing support to guide users to critique the current example product for comparing with other tradeoff alternatives

with other apartments for : This section shows the different factors that can be used for comparison. These include Better Type, Cheaper Price, Bigger Area, Better Bathroom, Better Kitchen, and Closer Distance. You are willing to compromise on the following attributes : This section allows users to indicate which of the factors they are willing to compromise on. In the image, there are no selections made.
Cancel : This button allows users to exit the comparison window.
Show Results : This button allows users to see the apartments that are most similar to Apt 34 based on the user's selected comparison criteria.
By using this comparison tool, users can efficiently identify apartments that meet their needs and preferences.

Critiquing-based recommenders: survey and emerging trends

Critiquing: Example

The image you sent depicts a critiquing interface from a recommender system specializing in digital cameras [1]. Critiquing recommender systems allow users to iteratively refine recommendations provided by the system based on their preferences.

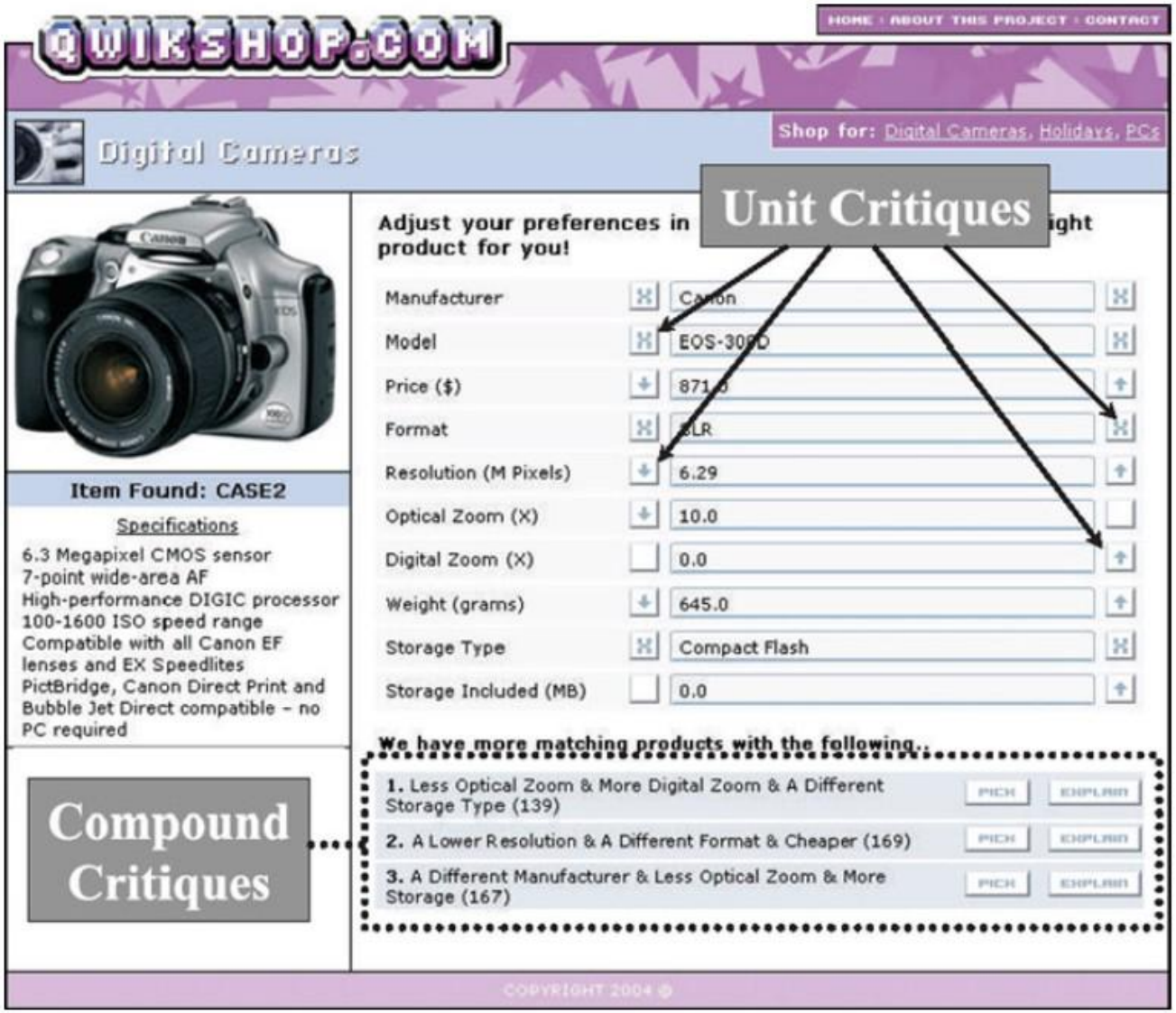


Fig. 5 The Dynamic Critiquing interface with system suggested compound critiques for users to select (McCarthy et al. 2005c)

Knowledge-based Recommendations: Limitations

Cost of Knowledge Acquisition:

Building and maintaining a comprehensive knowledge base can be expensive. It requires gathering data on users (preferences, demographics), items (features, descriptions) and the relationships between them.
Project Proposal Consideration: When proposing a KBRS project, consider the cost-effectiveness of knowledge acquisition. Is the value of the recommendations worth the investment in building and maintaining the knowledge base?

- cost of knowledge acquisition (consider your project proposals)
- accuracy of models
- independence assumption for preferences

Accuracy of Models:

The accuracy of KBRS recommendations hinges on the accuracy and completeness of the knowledge base. Incomplete or inaccurate data can lead to flawed recommendations.
Project Proposal Consideration: Develop strategies for ensuring the quality and ongoing maintenance of the knowledge base in your project proposal. This might involve outlining data collection methods and quality control procedures.

Independence Assumption for Preferences:

KBRS often assumes user preferences are independent of each other. In reality, preferences can be interconnected. For instance, a user who prefers high-resolution cameras might also prioritize large storage capacity.
Project Proposal Consideration: Explore techniques to model the relationships between user preferences within your project proposal. This could involve implementing more sophisticated recommendation algorithms that can capture these interdependencies.

Hybrid Methods

Hybridization combines these techniques to leverage their strengths and mitigate their weaknesses. Here's your example:
Example: CF + CBF (overcoming cold start problem):
A new user signs up for a music streaming service.
CF alone might struggle as there's no data on the user's preferences.
A hybrid system could use CBF to analyze basic information from the user profile (e.g., age, location) and recommend popular music genres enjoyed by users with similar demographics.
As the user interacts with the service (ratings, listens), CF can kick in and recommend more personalized music based on their evolving taste.

collaborative filtering: *"what is popular among my peers"*
content-based: *"more of the same"*
knowledge-based: *"what fits my needs"*

- each has advantages and disadvantages
- hybridization – combine more techniques, avoid some shortcomings
- simple example: CF with content-based (or simple "popularity recommendation") to overcome "cold start problem"

Here are some additional benefits of hybridization:

Improved Accuracy and Coverage: Combining multiple techniques can lead to more accurate and diverse recommendations.
Flexibility: Hybridization allows for customization based on the specific application and data available.
Overall, hybridization is a powerful approach to building robust and effective recommender systems.

Hybridization Designs

- monolitic desing, combining different
- features parallel use of several systems,
- weighting/voting pipelined invocation of different systems

refer pdf

Types of Recommender Systems

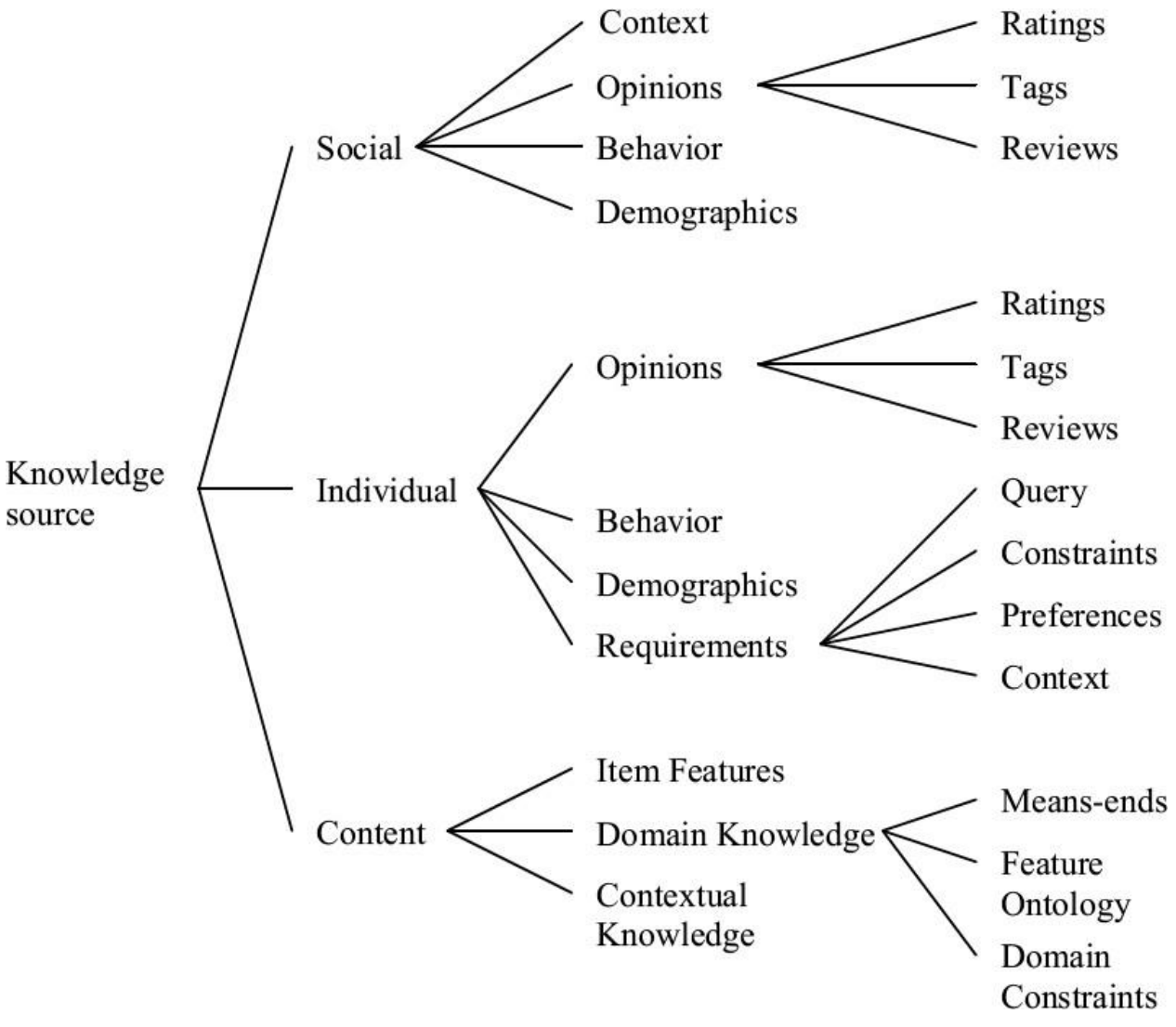
- non-personalized
- demographic
- collaborative
- filtering content
- based
- knowledge-based
- hybrid

what to apply when?

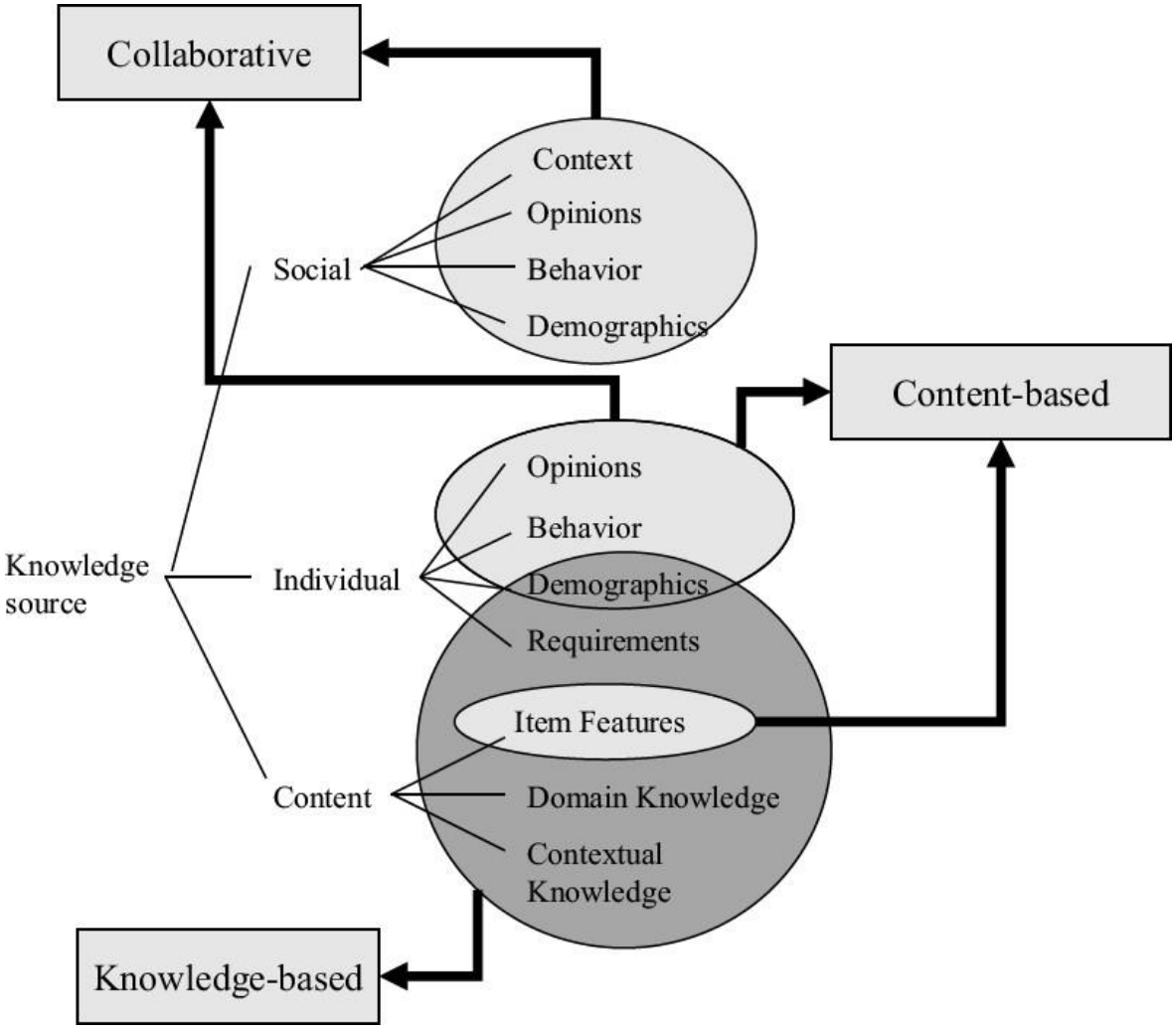
Non-personalized: General recommendations, good for new users (low data needed).
Demographic CF: Uses user demographics (age, location) for recommendations (effective when demographics predict preference).
Content-Based Filtering: Recommends similar items to what a user liked in the past (good for well-defined item attributes and aligned user preferences).
Knowledge-Based: Uses a knowledge base to recommend items based on user needs and item functionalities (ideal for complex relationships between items and needs).
Hybrid: Combines techniques to address limitations (e.g., CF + CBF for new users).
Choosing the right approach depends on data, domain, and goals.

Non-personalized for general browsing.
CF or CBF for personalization based on user data or item features.
KBRS for complex relationships and user needs.
Hybrid for most robust and personalized recommendations.

Taxonomy of Knowledge Sources



Knowledge Sources and Recommendation Types



Matching Recommendation Technologies and Domains

Sample Domains for Recommendation

Domain	Risk	Churn	Heterog-eneous	Preferences	Interaction Style	Scrutabi-lity	Examples	Technology
News	Low	High	Low	Stable?	Implicit	Not required	Yahoo news[6] ACR news[45] and [38] Google news[16]	Content-based Collaborative-Filtering
E-commerce	Low	High	High	Stable	Implicit	Not required	Amazon.com eBay	Collaborative-Filtering
Web Page Recom-mender	Low	High	High	Unstable	Implicit	Not required	[9, 36, 4]	Collaborative-Filtering Hybrid
Movie	Low	Low	Low	Stable	Implicit	Not required	Netflix[50, 64] Movielens[21]	Collaborative-Filtering
Music	Low	Low	Low	Stable?	Implicit	Not required	Pandora and [24, 28, 14]	Content-based Hybrid
Financial-services Life-insurance	High	Low	Low	Stable	Explicit	Required	Koba4MS[17] FSAdvisor[19] [65]	Knowledge-Based
Software Engineer-ing	Low	Low	Low	Stable	Explicit /Im-plicit	Required	[13] and [29]	Hybrid and Content-based
Tourism	High	Low	Low	Unstable	Explicit	Required	Travel Recom-mender [55] [37]	Content-based Knowledge-based
Job search Recruit-ing	High	Low	Low	Stable	Explicit	Required	CASPER [35] and [39]	Content-based
Real Estate	High	Low	Low	Stable	Explicit	Required	RentMe [10] FlatFinder[67] and [73]	Knowledge-based

Matching Recommendation Technologies and Domains

Explanations of Recommendations

- recommendations: selection (ranked list) of
- items explanations: (some) reasons for the choice

Goals of Providing Explanations

Why explanations?

Goals of Providing Explanations

Why explanations?

- transparency, trustworthiness, validity, satisfaction (users are more likely to use the system)
- persuasiveness (users are more likely to follow recommendations) effective
- effectiveness, efficiency (users can make better/faster decisions)
- education (users understand better the behaviour of the system, may use it in better ways)

Examples of Explanations

- knowledge-based recommenders
  - "Because you, as a customer, told us that simple handling of car is important to you, we included a special sensor system in our offer that will help you park your car easily."
  - algorithms based on CSP representation

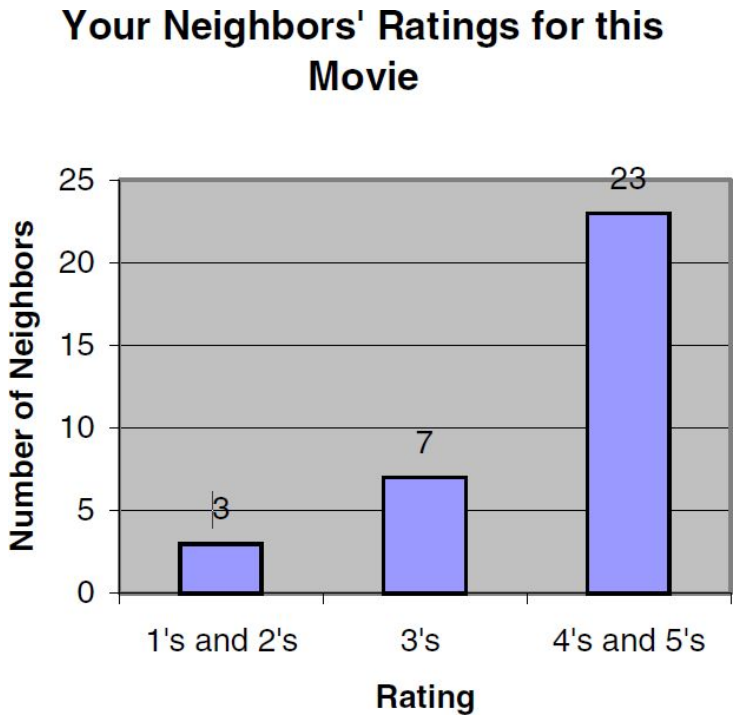
Examples of Explanations

- knowledge-based recommenders
  - "Because you, as a customer, told us that simple handling of car is important to you, we included a special sensor system in our offer that will help you park your car easily."
  - algorithms based on CSP representation
- recommendations based on item-similarity
  - "Because you watched X we recommend Y"

Explanations – Collaborative Filtering

Your Neighbors' Ratings for this Movie

Rating	Number of Neighbors
★	1
★★	2
★★★	7
★★★★	14
★★★★★	9



Explaining Collaborative Filtering Recommendations, Herlocker, Konstan, Riedl

Explanations – Collaborative Filtering

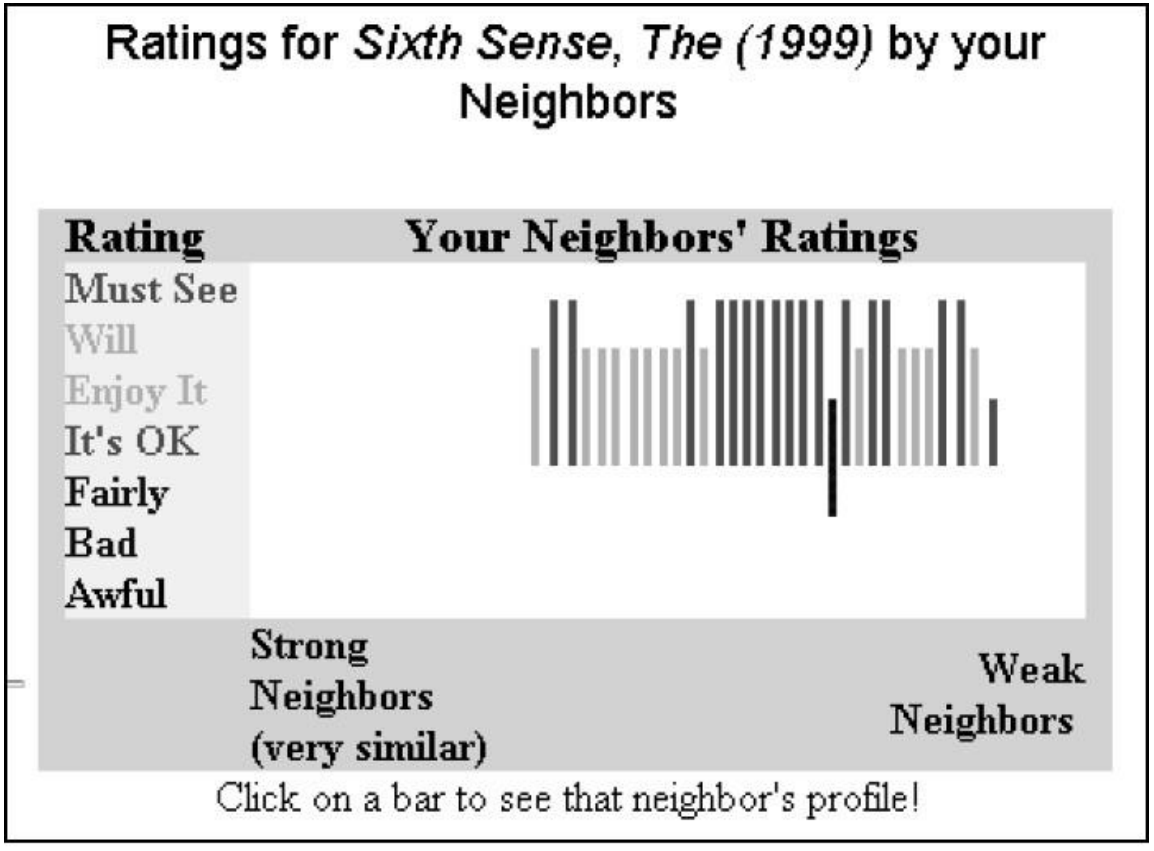


Figure 4. A screen explaining the recommendation for the movie "The Sixth Sense." Each bar represents a rating of a neighbor. Upwardly trending bars are positive ratings, while downward trending ones are negative. The x-axis represents similarity to the user.

Explaining Collaborative Filtering Recommendations, Herlocker, Konstan, Riedl

Explanations – Comparison

#		N	Mean Response	Std Dev
1	Histogram with grouping	76	5.25	1.29
2	Past performance	77	5.19	1.16
3	Neighbor ratings histogram	78	5.09	1.22
4	Table of neighbors ratings	78	4.97	1.29
5	Similarity to other movies rated	77	4.97	1.50
6	Favorite actor or actress	76	4.92	1.73
7	MovieLens percent confidence in prediction	77	4.71	1.02
8	Won awards	76	4.67	1.49
9	Detailed process description	77	4.64	1.40
10	# neighbors	75	4.60	1.29
11	No extra data – focus on system	75	4.53	1.20
12	No extra data – focus on users	78	4.51	1.35
13	MovieLens confidence in prediction	77	4.51	1.20
14	Good profile	77	4.45	1.53
15	Overall percent rated 4+	75	4.37	1.26
16	Complex graph: count, ratings, similarity	74	4.36	1.47
17	Recommended by movie critics	76	4.21	1.47
18	Rating and %agreement of closest neighbor	77	4.21	1.20
19	# neighbors with std. deviation	78	4.19	1.45
20	# neighbors with avg correlation	76	4.08	1.46
21	Overall average rating	77	3.94	1.22

Table 1. Mean response of users to each explanation interface, based on a scale of one to seven. Explanations 11 and 12 represent the base case of no additional information. Shaded rows indicate explanations with a mean response significantly different from the base cases (two-tailed $\alpha = 0.05$).

Moment of Recommendation

- front page,
- dashboard
- follow-up
- sidebar

on demand

recommendation placements may differ in their requirements

Your Projects: Questions

- What is the purpose / use case? What is the "business model"?
- What will you recommend? In what
- situation? A new system or extention of
- an existing one? **What data you have?**
  - items
  - user preferences; explicit/implicit ratings?
- Which techniques are relevant/suitable for you project? Collaborative filtering? Content-based? Knowledge-based? Combination?
- Are the following notions relevant: taxonomy, critiquing, explanations?