Artificial Intelligence Applied to the Web

Chapter 4 Part 5 - Recommendation Systems III

Diego Cornejo, Felipe Hernández and Juan Velásquez

University of Chile
Departament of Industrial Engineering

Spring 2025

Outline

Knowledge-Based Recommendation

Why Another Approach?

- Collaborative Filtering (CF) and Content-Based (CB) approaches are not always the best choice.
- They struggle with items that are purchased infrequently (like cars, houses, or computers) due to the lack of rating data.
- For these items, user preferences also evolve over time, making old ratings unreliable (e.g., five-year-old ratings for cellphones might be rather inappropriate now).
- Users in complex domains often want to define explicit requirements (e.g., “maximum price is X” or “car must be red”), which is not typical for pure CF or CB systems.

Knowledge-Based Recommendation

The Core Idea

- Knowledge-Based (KB) systems tackle these challenges by relying on explicit knowledge about the items and the user's needs.
- **Key Advantage:** They do not suffer from ramp-up or "cold-start" problems because they don't need user rating data to function.
- The recommendation process is highly interactive and often described as "conversational".

Knowledge-Based Recommendation

Two Basic Types

- KB systems calculate recommendations based on how items fit a user's requirements, not on community ratings.
- The two main types differ in how they use the provided knowledge:
  - **Constraint-based:** Relies on an explicitly defined set of recommendation rules and constraints to find matching products.
  - **Case-based:** Focuses on retrieving similar items using different types of similarity measures.
- In both approaches, the user specifies their needs, and the system tries to identify a solution.

Knowledge Representation

Constraints-based

- This approach models the recommendation problem as a **Constraint Satisfaction Problem (CSP)**.
- A recommendation is a **solution** that satisfies all active constraints.
- A CSP problem can be described by a-tuple (V, D, C) where:
  - V is a set of variables (item attributes).
  - D is a set of domains (possible values for each attribute).
  - C is a set of constraints (rules that define valid combinations of attribute values).

Knowledge Representation

Constraints-based

- Each solution to CSP can be described as:

$$\mathrm{CSP}(V = V_c \cup V_{\mathrm{PROD}}, D, C = C_R \cup C_F \cup C_{\mathrm{PROD}} \cup \mathrm{REQ})$$

- V_c : Customer properties. Describe the possible customer requirements.
- V_{PROD} : Product properties. Describe the product attributes.
- C_R : compatibility constraints. Define allowed instantiations of customer properties.
- C_F : Filter conditions. Define under which conditions which products should be selected.
- C_{PROD} : Product constraints. Define the currently available product assortment.

Knowledge Representation

Constraints-based 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

Knowledge Representation

Constraints-based Example

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

Knowledge Representation

Case-based

- In this approach, items are treated as "cases" and are retrieved using **similarity measures**.
- The system calculates how well each item's properties match the user's requirements.
- The overall similarity is a **weighted sum** of the similarities of individual attributes, where weights represent the importance of each requirement to the user.

Similarity Measures in Detail

Case-based

- Different attributes require different similarity calculations:
  - **More-is-Better (MIB):** For properties the user wants to maximize (e.g., camera resolution). Higher values are better.
  - **Less-is-Better (LIB):** For properties the user wants to minimize (e.g., price). Lower values are better.
  - **Distance-Based:** For when the user has a specific target value in mind (e.g., a specific monitor size), and closeness to that value is what matters most.

Interacting with Constraint-Based Recommenders

The General Interaction Flow

- The user interaction in a conversational, knowledge-based system typically follows these steps:
  - **1. Specify Preferences:** The user provides their initial requirements, often through a form or an interactive, wizard-style dialog.
  - **2. Get Recommendations:** Once enough information is gathered, the system presents a set of matching items.
  - **3. Revise Requirements:** The user can then change their requirements to see alternative solutions or to narrow down the number of matching items.

Interacting with Constraint-Based Recommenders

Defaults: Proposing Values

- Pre-filled or suggested values that support the user during the requirements process.
- Why are they useful?
  - They help customers who are unsure or lack technical knowledge by proposing a reasonable starting point.
  - They can reduce the effort needed to specify requirements.
- The main coin is that Defaults can also be used to bias or manipulate users into choosing certain options the seller wants to promote.

Interacting with Constraint-Based Recommenders

Defaults: Proposing Values

- Defaults can be specified in various ways:
  - **Static Defaults:** One fixed default is specified for each property (e.g., the default `usage` is `large-print`).
  - **Dependent Defaults:** The default value for one property depends on the user's choices for other properties (e.g., if `usage` is `small-print`, the default `max-price` becomes `300`).
  - **Derived Defaults:** The system automatically learns and proposes defaults by analyzing the interaction logs of previous users.

Interacting with Constraint-Based Recommenders

Defaults: Calculating Derived Defaults

- Derived defaults are often calculated using methods like:
  - **1-Nearest Neighbor:** The system finds the **single most similar past user** based on the current user's requirements and proposes their choice as the default.
  - **Weighted Majority Voter:** The system finds a set of similar past users (neighbors) and proposes the value that was **most frequently chosen** among them.
- The main problem is that these methods don't guarantee that a product will actually exist that matches the proposed default along with the user's other requirements.

Interacting with Constraint-Based Recommenders

Defaults: Selecting next question

- The same principles can be used to help the user decide which requirement to specify next.
- Instead of just waiting for input, the system can proactively suggest properties that might be interesting to the user.
- How it works:
  - The system can propose the most **popular** next question based on how frequently other users specified that property.
  - Alternatively, it can use the **weighted majority voter** to find what similar users specified next in their sessions.

Interacting with Constraint-Based Recommenders

The "No Solution Found" Dilemma

- This simple interaction flow is often not enough for practical applications.
- A common problem is when **no items in the catalog satisfy all of the user's requirements**.
- In these "no solution" situations, a good conversational recommender should intelligently support the user.
- The system should proactively help the user resolve the problem, for example, by proposing alternative actions to take.

Interacting with Constraint-Based Recommenders

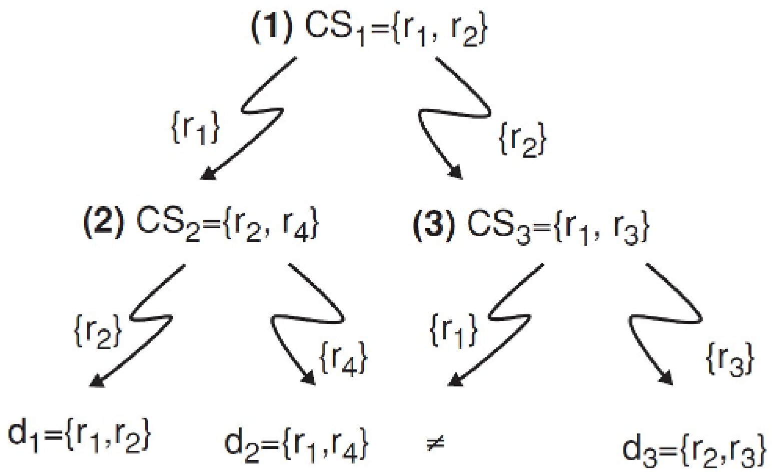
Unsatisfied requirements

- "No solution could be found"
- Constraint relaxation
  - The goal is to identify relaxations to the original set of constraints.
  - Relax constraints of a recommendation problem until a corresponding solution has been found.
- Users could also be interested in repair proposals.
  - Recommender can calculate a solution by adapting the proposed requirements.

Interacting with Constraint-Based Recommenders

Deal with unsatisfied requirements

- Calculate diagnoses for unsatisfied requirements.



- The diagnoses derived from the conflict sets $\{\mathrm{CS}_1, \mathrm{CS}_2, \mathrm{CS}_3\}$ are $\{d_1 : \{r_1, r_2\}, d_2 : \{r_1, r_4\}, d_3 : \{r_2, r_3\}\}$

Interacting with Constraint-Based Recommenders

Deal with unsatisfied requirements - QuickXPlain

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

Interacting with Constraint-Based Recommenders

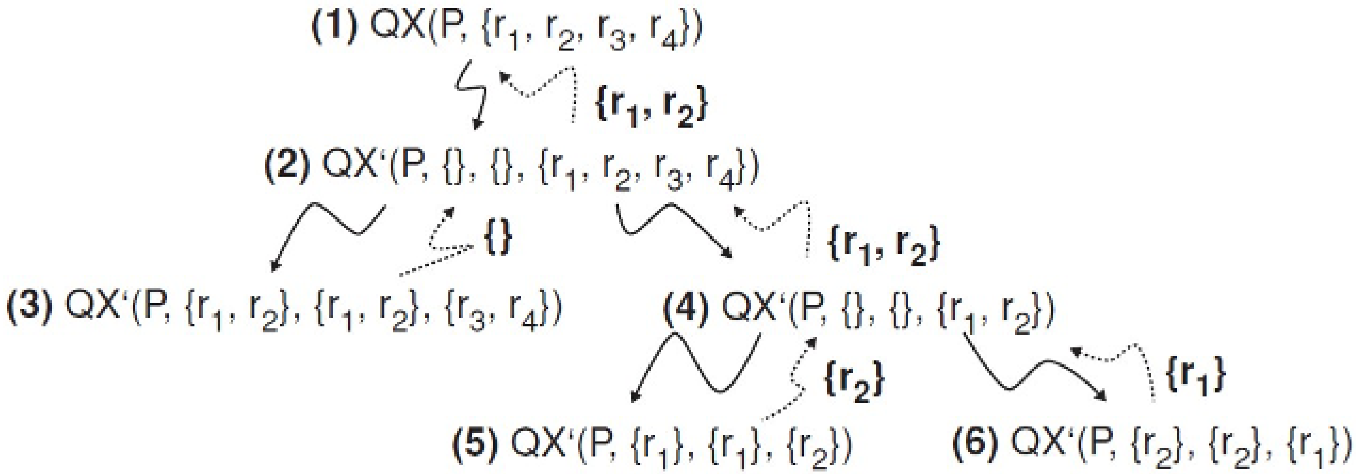
Deal with unsatisfied requirements - QuickXPlain

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

$$\text{REQ} = \{r_1 : \text{price} \leq 150, r_2 : \text{opt-zoom} = 5x, r_3 : \text{sound} = \text{yes}, r_4 : \text{waterproof} = \text{yes}\}$$

Interacting with Constraint-Based Recommenders

Deal with unsatisfied requirements - QuickXPlain



Interacting with Constraint-Based Recommenders

Repairs for unsatisfied requirements

- Identify possible adaptations.
- Or query the product table P with $\pi[\text{attributes}(d)]\sigma[\text{REQ} - d](P)$
  - $\pi[\text{attributes}(d_1)]\sigma[\text{REQ} - d_1](P) = \{\text{price} = 278, \text{opt-zoom} = 10x\}$
  - $\pi[\text{attributes}(d_2)]\sigma[\text{REQ} - d_2](P) = \{\text{price} = 182, \text{waterproof} = \text{no}\}$
  - $\pi[\text{attributes}(d_3)]\sigma[\text{REQ} - d_3](P) = \{\text{opt-zoom} = 4x, \text{sound} = \text{no}\}$

repair	price(€)	opt-zoom	sound	waterproof
Rep_1	278	10×	√	√
Rep_2	182	√	√	no
Rep_3	√	4×	no	√

Interacting with Constraint-Based Recommenders

Ranking the items

- Multi-attribute utility theory
  - Each item is evaluated according to a predefined set of dimensions that provide an aggregated view on the basic item properties.
- E.g. quality and economy are dimensions in the domain of digital cameras.

repair	price(€)	opt-zoom	sound	waterproof
Rep_1	278	10×	√	√
Rep_2	182	√	√	no
Rep_3	√	4×	no	√

Interacting with Constraint-Based Recommenders

Item utility for customers

- Ranking recommended items by their **utility** to the customer is crucial.
- This leverages the **primacy effect:** customers pay more attention to items at the top of a list.
- A good ranking, therefore, can significantly increase a user's **trust** and their **willingness to buy**.
- In KB systems, this is often done using **Multi-Attribute Utility Theory (MAUT)**, which evaluates the specific utility of each item for the customer.

Interacting with Constraint-Based Recommenders

Item utility for customers

- Example:

	value	quality	economy
price	$\leq$250	5	10
	$>$250	10	5
mpix	$\leq$8	4	10
	$>$8	10	6
opt-zoom	$\leq$9	6	9
	$>$9	10	6
LCD-size	$\leq$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

Interacting with Constraint-Based Recommenders

Item utility for customers

- Customer specific interest.

Customer	quality	economy
Cu_1	80%	20%
Cu_2	40%	60%

Interacting with Constraint-Based Recommenders

Item utility for customers

- Calculation of Utility

quality	economy	cu_1	cu_2
P_1 Σ(5,4,6,6,3,7,10) = 41	Σ (10,10,9,10,10,10,6) = 65	45.8 [8]	55.4 [6]
P_2 Σ(5,4,6,6,10,10,8) = 49	Σ (10,10,9,10,7,8,10) = 64	52.0 [7]	58.0 [1]
P_3 Σ(5,4,10,6,10,10,8) = 53	Σ (10,10,6,10,7,8,10) = 61	54.6 [5]	57.8 [2]
P_4 Σ(5,10,10,6,10,7,10) = 58	Σ (10,6,6,10,7,10,6) = 55	57.4 [4]	56.2 [4]
P_5 Σ(5,4,6,10,10,10,8) = 53	Σ (10,10,9,6,7,8,10) = 60	54.4 [6]	57.2 [3]
P_6 Σ(5,10,6,9,10,10,8) = 58	Σ (10,6,9,5,7,8,10) = 55	57.4 [3]	56.2 [5]
P_7 Σ(10,10,6,9,10,10,8) = 63	Σ (5,6,9,5,7,8,10) = 50	60.4 [2]	55.2 [7]
P_8 Σ(10,10,10,9,10,10,10) = 69	Σ (5,6,6,5,7,8,6) = 43	63.8 [1]	53.4 [8]

Interacting with Case-Based Recommenders

Case-based recommender systems

- Items are retrieved using similarity measures
- Distance similarity

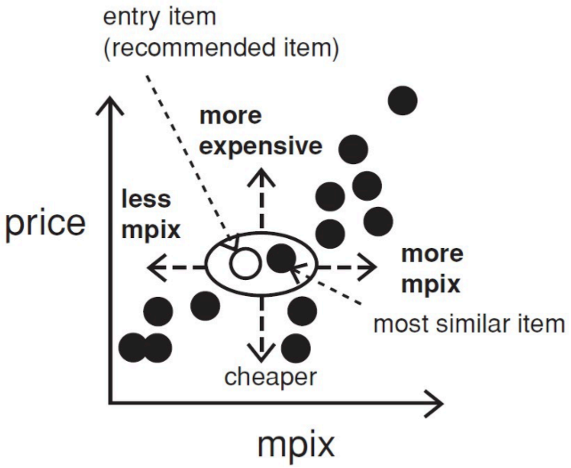
$$\text{similarity}(p, \text{REQ}) = \frac{\sum_{r \in \text{ REQ}} w_r \times \text{sim}(p, r)}{\sum_{r \in \text{ REQ}} w_r}$$

- Where
  - $\text{sim}(p, r)$ expresses for each item attribute value $\phi(p)$ its distance to the customer requirement $r \in \text{REQ}$.
  - w_r is the importance weight for requirement r

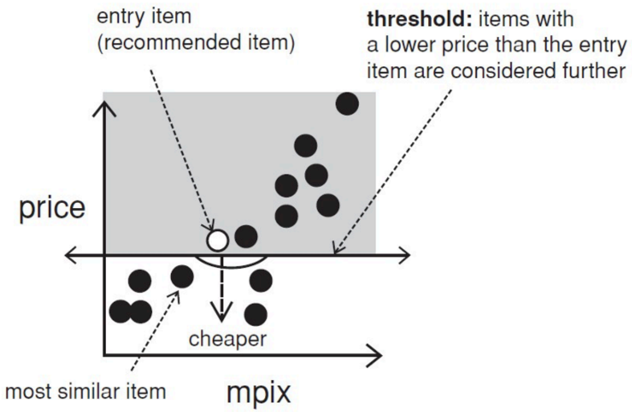
Interacting with Case-Based Recommenders

Interacting with case-based recommenders

- Customers maybe not know what they are seeking.
- Critiquing is an effective way to support such navigations.
- Customers specify their change requests (price or mpix) that are not satisfied by the current item (entry item).



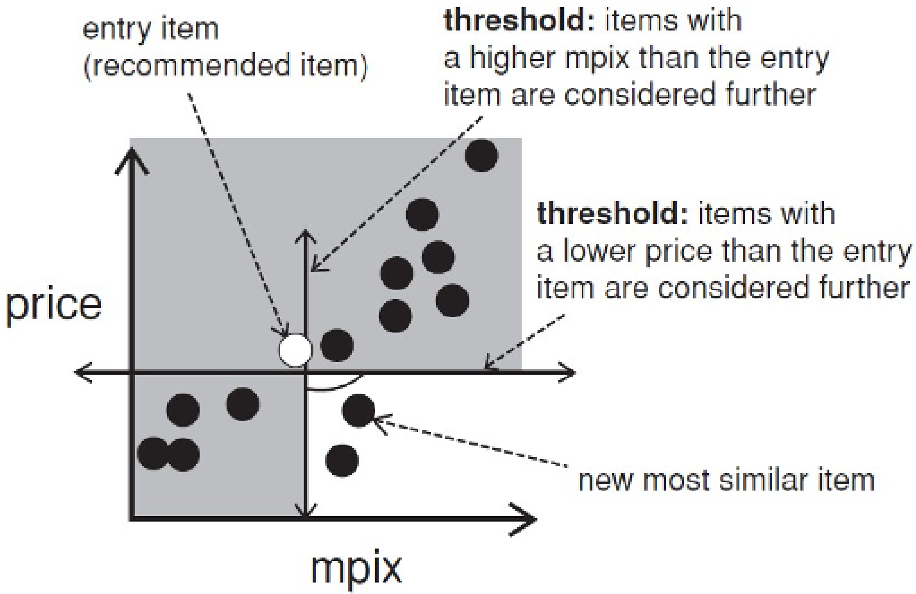
Critique on price



Interacting with Case-Based Recommenders

Compound critiques

- Operate over multiple properties can improve the efficiency of recommendation dialogs



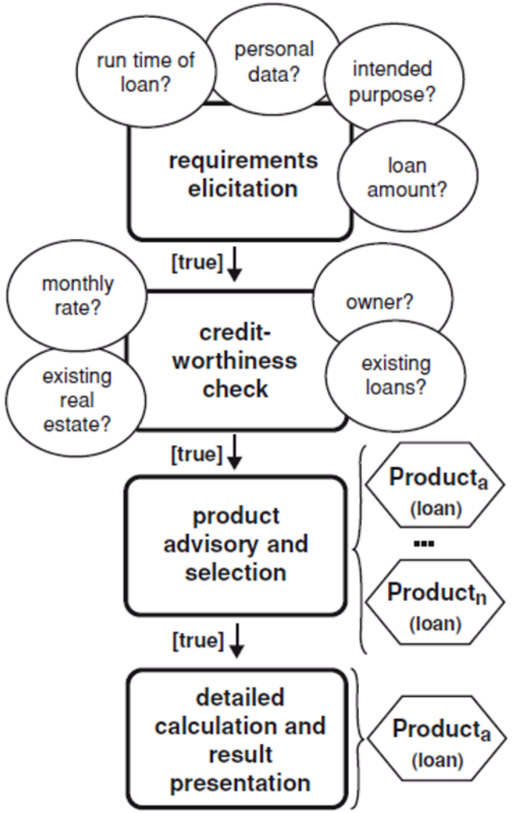
Interacting with Case-Based Recommenders

Dynamic critiques

- Exploits patterns, which are generic descriptions of differences between the recommended (entry) and the candidate items.
- Denoted as dynamic because they are derived on the fly in each critiquing cycle.
- Are calculated using the concept of association rule mining.
- Example:
  - "42.9% of the remaining digital cameras have a higher zoom and a lower price" (more zoom and lower price).

Interacting with Case-Based Recommenders

Example: Sales dialogue financial services



- In the financial services domain
  - Sales representatives do not know which services should be recommended
  - Improve the overall productivity of sales representatives
- Resembles call-center scripting
  - Best-practice sales dialogues
  - States, transitions with predicates
- Research results
  - Support for KA and validation
  - Node properties (reachable, extensible, deterministic)

Artificial Intelligence Applied to the Web

Chapter 4 Part 5 - Recommendation Systems III

Diego Cornejo, Felipe Hernández and Juan Velásquez

University of Chile
Departament of Industrial Engineering

Spring 2025