Knowledge-based recommendation
Basic I/O Relationship

Knowledge-based: "Tell me what fits based on my needs"
Why do we need knowledge-based recommendation?

- Products with low number of available ratings

- Time span plays an important role
  - five-year-old ratings for computers
  - user lifestyle or family situation changes

- Customers want to define their requirements explicitly
  - "the color of the car should be black"
Knowledge-based recommender systems

- **Constraint-based**
  - based on explicitly defined set of recommendation rules
  - fulfill recommendation rules

- **Case-based**
  - based on different types of similarity measures
  - retrieve items that are similar to specified requirements

- **Both approaches are similar in their conversational recommendation process**
  - users specify the requirements
  - systems try to identify solutions
  - if no solution can be found, users change requirements
Constraint-based recommender systems

- **Knowledge base**
  - usually mediates between user model and item properties
  - variables
    - user model features (requirements), Item features (catalogue)
  - set of constraints
    - logical implications (IF user requires A THEN proposed item should possess feature B)
    - hard and soft/weighted constraints
    - solution preferences

- **Derive a set of recommendable items**
  - fulfilling set of applicable constraints
  - applicability of constraints depends on current user model
  - explanations – transparent line of reasoning
Constraint-based recommendation tasks

- Find a set of user requirements such that a subset of items fulfills all constraints
  - ask user which requirements should be relaxed/modified such that some items exist that do not violate any constraint

- Find a subset of items that satisfy the maximum set of weighted constraints
  - similar to find a maximally succeeding subquery (XSS)
  - all proposed items have to fulfill the same set of constraints
  - compute relaxations based on predetermined weights

- Rank items according to weights of satisfied soft constraints
  - rank items based on the ratio of fulfilled constraints
  - does not require additional ranking scheme
Constraint-based recommendation problem

- Select items from this catalog that match the user's requirements

<table>
<thead>
<tr>
<th>id</th>
<th>price(€)</th>
<th>mpix</th>
<th>opt-zoom</th>
<th>LCD-size</th>
<th>movies</th>
<th>sound</th>
<th>waterproof</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>148</td>
<td>8.0</td>
<td>4×</td>
<td>2.5</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>P₂</td>
<td>182</td>
<td>8.0</td>
<td>5×</td>
<td>2.7</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>P₃</td>
<td>189</td>
<td>8.0</td>
<td>10×</td>
<td>2.5</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>P₄</td>
<td>196</td>
<td>10.0</td>
<td>12×</td>
<td>2.7</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>P₅</td>
<td>151</td>
<td>7.1</td>
<td>3×</td>
<td>3.0</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>P₆</td>
<td>199</td>
<td>9.0</td>
<td>3×</td>
<td>3.0</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>P₇</td>
<td>259</td>
<td>10.0</td>
<td>3×</td>
<td>3.0</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>P₈</td>
<td>278</td>
<td>9.1</td>
<td>10×</td>
<td>3.0</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

- User's requirements can, for example, be
  - "the price should be lower than 300 €"
  - "the camera should be suited for sports photography"
Constraint satisfaction problem (CSP)

- A knowledge-based RS with declarative knowledge representation
  \[ \text{CSP} \left( X_I \cup X_U, D, SRS \cup KB \cup I \right) \]

- Def.
  - \( X_I, X_U \): Variables describing product and user model with domain \( D \)
  - \( KB \): Knowledge base with domain restrictions (e.g. if purpose=\textit{on travel} then lower focal length < \( 28\text{mm} \))
  - \( SRS \): Specific requirements of user (e.g. purpose = \textit{on travel})
  - \( I \): Product catalog

- Solution: Assignment tuple
  \[ \theta \ \forall x \in X_I (x = v) \in \theta \land v \in \text{dom}(x) \]
  \[ \text{s.t.} SRS \cup KB \cup I \cup \theta \text{ is satisfiable} \]
Conjunctive query

- **Different from a constraint solver**
  - it is not to find valid instantiations for a CSP

- **Conjunctive query is executed in the item catalog**
  - a conjunctive database query
  - a set of selection criteria that are connected conjunctively

- **σ[criteria](P)**
  - $P$: product assortment
  - example: $σ[mpix≥10, price<300](P) = \{p4, p7\}$
Interacting with constraint-based recommenders

- The user specifies his or her initial preferences
  - all at once or
  - incrementally in a wizard-style
  - interactive dialog

- The user is presented with a set of matching items
  - with explanation as to why a certain item was recommended

- The user might revise his or her requirements
  - see alternative solutions
  - narrow down the number of matching items
Defaults

- **Support customers to choose a reasonable alternative**
  - unsure about which option to select
  - simply do not know technical details

- **Type of defaults**
  - static defaults
  - dependent defaults
  - derived defaults

- **Selecting the next question**
  - most users are not interested in specifying values for all properties
  - identify properties that may be interesting for the user
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
Deal with unsatisfied requirements

- Calculate diagnoses for unsatisfied requirements

\[\begin{align*}
\text{(1) } & CS_1 = \{r_1, r_2\} \\
\text{(2) } & CS_2 = \{r_2, r_4\} \\
\text{(3) } & CS_3 = \{r_1, r_3\} \\
\end{align*}\]

- The diagnoses derived from the conflict sets \{CS_1, CS_2, CS_3\} are \(d_1 = \{r_1, r_2\}, d_2 = \{r_1, r_4\}, d_3 = \{r_2, r_3\}\)
QuickXPlain

- **Calculate conflict sets**

  Algorithm 4.1 QuickXPlain\((P, REQ)\)

  **Input:** trusted knowledge (items) \(P\); Set of requirements \(REQ\)
  **Output:** minimal conflict set \(CS\)

  if \(\sigma_{[REQ]}(P) = \emptyset\) or \(REQ = \emptyset\) then return \(\emptyset\)
  else return \(QX'(P, \emptyset, \emptyset, REQ)\);

  Function \(QX'(P, B, \Delta, REQ)\)

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

  return \(\Delta_1 \cup \Delta_2\);
Example of QuickXPlain

<table>
<thead>
<tr>
<th>id</th>
<th>Price(€)</th>
<th>mpix</th>
<th>opt-zoom</th>
<th>LCD-size</th>
<th>movies</th>
<th>sound</th>
<th>waterproof</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>148</td>
<td>8.0</td>
<td>4×</td>
<td>2.5</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>P₂</td>
<td>182</td>
<td>8.0</td>
<td>5×</td>
<td>2.7</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>P₃</td>
<td>189</td>
<td>8.0</td>
<td>10×</td>
<td>2.5</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>P₄</td>
<td>196</td>
<td>10.0</td>
<td>12×</td>
<td>2.7</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>P₅</td>
<td>151</td>
<td>7.1</td>
<td>3×</td>
<td>3.0</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>P₆</td>
<td>199</td>
<td>9.0</td>
<td>3×</td>
<td>3.0</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>P₇</td>
<td>259</td>
<td>10.0</td>
<td>3×</td>
<td>3.0</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>P₈</td>
<td>278</td>
<td>9.1</td>
<td>10×</td>
<td>3.0</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

- REQ = \{r1:price \leq 150, r2:opt-zoom = 5x, r3:sound = yes, r4:waterproof = yes\}

\[
\begin{align*}
(1) & \quad QX(P, \{r_1, r_2, r_3, r_4\}) \\
(2) & \quad QX'(P, \{\}, \{\}, \{r_1, r_2, r_3, r_4\}) \\
(3) & \quad QX'(P, \{r_1, r_2\}, \{r_1, r_2\}, \{r_3, r_4\}) \\
(4) & \quad QX'(P, \{\}, \{\}, \{r_1, r_2\}) \\
(5) & \quad QX'(P, \{r_1\}, \{r_1\}, \{r_2\}) \\
(6) & \quad QX'(P, \{r_2\}, \{r_2\}, \{r_1\})
\end{align*}
\]
Repairs for unsatisfied requirements

- Identify possible adaptations

- Or query the product table $P$ with $\pi[\text{attributes}(d)]\sigma[\text{REQ} \rightarrow d](P)$
  - $\pi[\text{attributes}(d1)]\sigma[\text{REQ} \rightarrow d1](P) = \{\text{price}=278, \text{opt}-\text{zoom}=10\times\}$
  - $\pi[\text{attributes}(d2)]\sigma[\text{REQ} \rightarrow d2](P) = \{\text{price}=182, \text{waterproof}=\text{no}\}$
  - $\pi[\text{attributes}(d3)]\sigma[\text{REQ} \rightarrow d3](P) = \{\text{opt}-\text{zoom}=4\times, \text{sound}=\text{no}\}$

<table>
<thead>
<tr>
<th>repair</th>
<th>price(€)</th>
<th>opt-zoom</th>
<th>sound</th>
<th>waterproof</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rep₁</td>
<td>278</td>
<td>10×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Rep₂</td>
<td>182</td>
<td>✓</td>
<td>✓</td>
<td>no</td>
</tr>
<tr>
<td>Rep₃</td>
<td>✓</td>
<td>4×</td>
<td>no</td>
<td>✓</td>
</tr>
</tbody>
</table>
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*

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
<th>quality</th>
<th>economy</th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>≤250</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>&gt;250</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>mpix</td>
<td>≤8</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>&gt;8</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>opt-zoom</td>
<td>≤9</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>&gt;9</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>LCD-size</td>
<td>≤2.7</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>&gt;2.7</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>movies</td>
<td>Yes</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>sound</td>
<td>Yes</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>waterproof</td>
<td>Yes</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>
Item utility for customers

- Customer specific interest

<table>
<thead>
<tr>
<th>Customer</th>
<th>quality</th>
<th>economy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cu₁</td>
<td>80%</td>
<td>20%</td>
</tr>
<tr>
<td>Cu₂</td>
<td>40%</td>
<td>60%</td>
</tr>
</tbody>
</table>

- Calculation of Utility

<table>
<thead>
<tr>
<th>quality</th>
<th>economy</th>
<th>cu₁</th>
<th>cu₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁ (\Sigma(5,4,6,6,3,7,10) = 41)</td>
<td>(\Sigma(10,10,9,10,10,10,6) = 65)</td>
<td>45.8 [8]</td>
<td>55.4 [6]</td>
</tr>
<tr>
<td>P₂ (\Sigma(5,4,6,6,10,10,8) = 49)</td>
<td>(\Sigma(10,10,9,10,7,8,10) = 64)</td>
<td>52.0 [7]</td>
<td>58.0 [1]</td>
</tr>
<tr>
<td>P₃ (\Sigma(5,4,10,6,10,10,8) = 53)</td>
<td>(\Sigma(10,10,6,10,7,8,10) = 61)</td>
<td>54.6 [5]</td>
<td>57.8 [2]</td>
</tr>
<tr>
<td>P₄ (\Sigma(5,10,10,6,10,7,10) = 58)</td>
<td>(\Sigma(10,6,6,10,7,10,6) = 55)</td>
<td>57.4 [4]</td>
<td>56.2 [4]</td>
</tr>
<tr>
<td>P₅ (\Sigma(5,4,6,10,10,10,8) = 53)</td>
<td>(\Sigma(10,10,9,6,7,8,10) = 60)</td>
<td>54.4 [6]</td>
<td>57.2 [3]</td>
</tr>
<tr>
<td>P₆ (\Sigma(5,10,6,9,10,10,8) = 58)</td>
<td>(\Sigma(10,6,9,5,7,8,10) = 55)</td>
<td>57.4 [3]</td>
<td>56.2 [5]</td>
</tr>
<tr>
<td>P₇ (\Sigma(10,10,6,9,10,10,8) = 63)</td>
<td>(\Sigma(5,6,9,5,7,8,10) = 50)</td>
<td>60.4 [2]</td>
<td>55.2 [7]</td>
</tr>
<tr>
<td>P₈ (\Sigma(10,10,10,9,10,10,10) = 69)</td>
<td>(\Sigma(5,6,6,5,7,8,6) = 43)</td>
<td>63.8 [1]</td>
<td>53.4 [8]</td>
</tr>
</tbody>
</table>
Case-based recommender systems

- Items are retrieved using similarity measures
- Distance similarity

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

- Def.
  - \( sim(p, r) \) expresses for each item attribute value \( \phi_r(p) \) its distance to the customer requirement \( r \in REQ \).
  - \( w_r \) is the importance weight for requirement \( r \)

- In real world, customer would like to
  - maximize certain properties. i.e. resolution of a camera, "more is better" (MIB)
  - minimize certain properties. i.e. price of a camera, "less is better" (LIB)
Interacting with case-based recommenders

- Customers may 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)*
Compound critiques

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

- **Association rule mining**

- **Basic steps for dynamic critiques**
  - *q*: initial set of requirements
  - *CI*: all the available items
  - *K*: maximum number of compound critiques
  - *σ_min*: minimum support value for calculated association rules.

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 *σ_min*

procedure DynamicCritiquing(*q, CI, k, σ_min*)
repeat
  *r* ← ItemRecommend(*q, CI*);
  *CC* ← CompoundCritiques(*r, CI, k, σ_min*);
  *q* ← UserReview(*r, CI, CC*);
until empty(*q*)
end procedure

procedure ItemRecommend(*q, CI*)
*CI* ← \{ci ∈ *CI*: satisfies(ci, *q*)};
*r* ← mostsimilar(*CI, q*);
return *r*;
end procedure

procedure UserReview(*r, CI, CC*)
*q* ← critique(*r, CC*);
*CI* ← *CI* − *r*;
return *q*;
end procedure

procedure CompoundCritiques(*r, CI, k, σ_min*)
*CP* ← CritiquePatterns(*r, CI*);
*CC* ← Apriori(*CP, σ_min*);
*SC* ← SelectCritiques(*CC, k*);
return *SC*;
end procedure
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)
Example software: VITA sales support
Example: Critiquing

- Similarity-based navigation in item space
- Compound critiques
  - more efficient navigation than with unit critiques
  - mining of frequent patterns
- Dynamic critiques
  - only applicable compound critiques proposed
- Incremental critiques
  - considers history
- Adaptive suggestions
  - suggest items that allow to best refine user’s preference model
Summary

- **Knowledge-based recommender systems**
  - constraint-based
  - case-based

- **Limitations**
  - cost of knowledge acquisition
    - from domain experts
    - from users
    - from web resources
  - accuracy of preference models
    - very fine granular preference models require many interaction cycles
    - collaborative filtering models preference implicitly
  - independence assumption can be challenged
    - preferences are not always independent from each other