Learning Similarity Queries from Preferences

Ingo Schmitt

Brandenburgische Technische Universität Cottbus
Institut für Informatik
Lehrstuhl Datenbank- und Informationssysteme

December 15th, 2010
Outline

- Motivation
- Commuting Quantum Query Language
- Weighting of query conditions
- Quantitative preferences
- ForMat-Project
Motivating Example

Multimedia search: bridging the semantic gap

Find all paintings from Van Gogh using sample images, selected low-level-features, and appropriate similarity conditions

other scenarios: Skyline, Conjoint Analysis, Recommender Systems, ...
Questions to be answered

1. How to compose a complex similarity condition using a small set of generic similarity conditions?
Questions to be answered

1. How to compose a complex similarity condition using a small set of generic similarity conditions?
   ⇝ logic
Questions to be answered

1. How to compose a complex similarity condition using a small set of generic similarity conditions?
   \( \rightarrow \) logic

2. How to weight similarity conditions against each other?
Questions to be answered

1. How to compose a complex similarity condition using a small set of generic similarity conditions?
   ⇝ logic

2. How to weight similarity conditions against each other?
   ⇝ logic-based weighting
Questions to be answered

1. How to compose a complex similarity condition using a small set of generic similarity conditions?
   \(~\Rightarrow\ \text{logic}~

2. How to weight similarity conditions against each other?
   \(~\Rightarrow\ \text{logic-based weighting}~

3. Where does the weights come from?
Questions to be answered

1. How to compose a complex similarity condition using a small set of generic similarity conditions?
   \[\sim\ logic\]

2. How to weight similarity conditions against each other?
   \[\sim\ logic-based\ weighting\]

3. Where does the weights come from?
   \[\sim\ learning\ from\ qualitative\ and\ quantitative\ preferences\]
Questions to be answered

1. How to compose a complex similarity condition using a small set of generic similarity conditions?
   ⇝ logic

2. How to weight similarity conditions against each other?
   ⇝ logic-based weighting

3. Where does the weights come from?
   ⇝ learning from qualitative and quantitative preferences

4. How to learn a logical combination?
Questions to be answered

1. How to compose a complex similarity condition using a small set of generic similarity conditions?
   ⇒ logic

2. How to weight similarity conditions against each other?
   ⇒ logic-based weighting

3. Where does the weights come from?
   ⇒ learning from qualitative and quantitative preferences

4. How to learn a logical combination?
   ⇒ simplex algorithm
Main idea

- Combination of
  - boolean conditions: painter='van Gogh' ⇝ \{0, 1\} and
  - proximity conditions: texture ≈ c_{texture} ⇝ [0, 1]

within a logic: \( \land, \lor, \neg, \exists, \forall \), ...
Main idea

- Combination of
  - boolean conditions: \text{painter='van Gogh'} \sim \{0, 1\} and
  - proximity conditions: \text{texture} \sim c_{\text{texture}} \sim [0, 1]

within a logic: \land, \lor, \neg, \exists, \forall, \ldots

- Evaluation based on a well-founded theory
  \sim quantum mechanics and quantum logic
Main idea

- Combination of
  - boolean conditions: `painter='van Gogh' \sim \{0, 1\}` and
  - proximity conditions: `texture \sim c_{texture} \sim [0, 1]`

  within a logic: \(\land, \lor, \neg, \exists, \forall, \ldots\)

- Evaluation based on a well-founded theory
  \(\sim\) quantum mechanics and quantum logic

- Evaluation
  \(\sim\) logical transformations + simple arithmetic calculations
Commuting Quantum Query Language

- CQQL: Commuting Quantum Query Language
- extends the relational domain calculus with similarity predicates
- based conceptually on vector space model of quantum mechanics and logic
CQQL: Syntax

- analogous to relational domain calculus
- adds similarity predicate: ∼
CQQL: Syntax

- analogous to relational domain calculus
- adds similarity predicate: ~

Example query:

\[
\{ \text{Pid}\mid \exists \text{Painter, Texture, Paint\textunderscore technique} : \\
\text{Painting}(\text{Pid, Painter, Texture, Paint\textunderscore technique}) \land \\
\text{Painter} = \text{'van Gogh'} \land \text{Texture} \sim t_{\text{van Gogh}} \land \\
\text{Paint\textunderscore technique} \sim m_{\text{van Gogh}}\}
\]
Commuting conditions

- Condition set is **commuting**, if all atomic conditions represent orthonormal base vectors.
Commuting conditions

- condition set is commuting, if all atomic conditions represent orthonormal base vectors
  \( \Leftrightarrow \) for each attribute at most one similarity predicate
Commuting conditions

- Condition set is commuting, if all atomic conditions represent orthonormal base vectors.
  \[\sim\] for each attribute at most one similarity predicate.

- Quantum logic on commuting conditions forms a Boolean algebra.
Arithmetic evaluation of CQQL-queries

let $\varphi$ be a complex CQQL-condition constructed by applying $\land$, $\lor$, $\neg$ on atomic conditions and let $o$ be a database object

- Evaluation denoted by $f_\varphi(o)$
Arithmetic evaluation of CQQL-queries

let $\varphi$ be a complex CQQL-condition constructed by applying $\land$, $\lor$, $\neg$ on atomic conditions and let $o$ be a database object

- Evaluation denoted by $f_{\varphi}(o)$
- Evaluating an atomic condition: positive semi-definite correlation matrix
Arithmetic evaluation of CQQL-queries

let $\varphi$ be a complex CQQL-condition constructed by applying $\land, \lor, \neg$ on atomic conditions and let $o$ be a database object

- Evaluation denoted by $f_\varphi(o)$
- Evaluating an atomic condition:
  positive semi-definite correlation matrix
- Evaluation of negation: $f_{\neg \varphi}(o) = 1 - f_\varphi(o)$
Arithmetic evaluation of CQQL-queries

let \( \varphi \) be a complex CQQL-condition constructed by applying \( \land, \lor, \neg \) on atomic conditions and let \( o \) be a database object

- Evaluation denoted by \( f_{\varphi}(o) \)
- Evaluating an atomic condition: positive semi-definite correlation matrix
- Evaluation of negation: \( f_{\neg \varphi}(o) = 1 - f_{\varphi}(o) \)
- Evaluation of disjunction and conjunction: requires a certain syntactical form (normal form)
Conjunction and disjunction in CQQL

Evaluating a normalized CQQL-condition

- \( f_{\varphi_1 \land \varphi_2}(o) = f_{\varphi_1}(o) \times f_{\varphi_2}(o) \)
- \( f_{\varphi_1 \lor \varphi_2}(o) = f_{\varphi_1}(o) + f_{\varphi_2}(o) - f_{\varphi_1}(o) \times f_{\varphi_2}(o) \)
- \( f(\phi \land \varphi_1) \lor (\neg \phi \land \varphi_2)(o) = f_{\phi \land \varphi_1}(o) + f_{\neg \phi \land \varphi_2}(o) \)
Conjunction and disjunction in CQQL

Evaluating a normalized CQQL-condition

- \( f_{\varphi_1 \land \varphi_2}(o) = f_{\varphi_1}(o) \times f_{\varphi_2}(o) \)
- \( f_{\varphi_1 \lor \varphi_2}(o) = f_{\varphi_1}(o) + f_{\varphi_2}(o) - f_{\varphi_1}(o) \times f_{\varphi_2}(o) \)
- \( f_{(\phi \land \varphi_1) \lor (\neg \phi \land \varphi_2)}(o) = f_{\phi \land \varphi_1}(o) + f_{\neg \phi \land \varphi_2}(o) \)

Boolean algebra

\( \iff \) every complex condition can be normalized
Weighting

Hotel example: Prerow, close to beach, not expensive, family

Query tree

(blue for proximity condition)
# Hotel examples

<table>
<thead>
<tr>
<th>characteristic</th>
<th>Prerow</th>
<th>price</th>
<th>beach</th>
<th>family</th>
</tr>
</thead>
<tbody>
<tr>
<td>beach</td>
<td>1</td>
<td>0.6</td>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>family</td>
<td>1</td>
<td>0.6</td>
<td>0.1</td>
<td>1.0</td>
</tr>
<tr>
<td>cheap</td>
<td>1</td>
<td>0.9</td>
<td>0.4</td>
<td>0.5</td>
</tr>
<tr>
<td>good</td>
<td>1</td>
<td>0.65</td>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>too_expensive</td>
<td>1</td>
<td>0.1</td>
<td>0.4</td>
<td>0.5</td>
</tr>
<tr>
<td>bad</td>
<td>1</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
</tr>
</tbody>
</table>
Logic-based weighting

Weights $\implies$ logic

$$\varphi_1 \land \theta_1, \theta_2 \varphi_2 = (\varphi_1 \lor \neg \theta_1) \land (\varphi_2 \lor \neg \theta_2)$$

$$\varphi_1 \lor \theta_1, \theta_2 \varphi_2 = (\varphi_1 \land \theta_1) \lor (\varphi_2 \land \theta_2)$$
Weights in our example

\[ \neg \theta_{bf} \lor \theta_b \land \theta_f \land \text{family} \land \text{beach} \land \text{price} \land \text{Prerow} \]
## Example weights

### Results of executing weighted query

<table>
<thead>
<tr>
<th></th>
<th>$\theta_{bf}$</th>
<th>$\theta_b$</th>
<th>$\theta_f$</th>
<th>winner</th>
</tr>
</thead>
<tbody>
<tr>
<td>equal weights</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>good</td>
</tr>
<tr>
<td>only price</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>cheap</td>
</tr>
<tr>
<td>neglecting family</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>beach</td>
</tr>
<tr>
<td>neglecting beach</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>family</td>
</tr>
</tbody>
</table>
User is unable to specify weight values
example: specify a given color by adjusting rgb-sliders
Weight learning

- User is unable to specify weight values
  example: specify a given color by adjusting rgb-sliders
- Better: user sees, defines and corrects exclusively preferences
Weight learning

- User is unable to specify weight values
  example: specify a given color by adjusting rgb-sliders

- Better: user **sees, defines and corrects exclusively preferences**
  example: Hotel1 is better than Hotel2
Weight learning

- User is unable to specify weight values
  example: specify a given color by adjusting rgb-sliders

- Better: user sees, defines and corrects exclusively preferences
  example: Hotel1 is better than Hotel2

- Problem: how to map between preferences and weight values
Weight learning

- User is unable to specify weight values
  example: specify a given color by adjusting rgb-sliders

- Better: user *sees, defines and corrects exclusively* preferences
  example: Hotel1 is better than Hotel2

- Problem: how to *map* between preferences and weight values

- requires query iteration $\leadsto$ Relevance Feedback
Query refinement

- User interactions
  - poset
  - poset’
  - poset’´

- Rank
  - Initial weights

- Weights’
  - Query evaluation
  - Reduction
  - Modification
  - Learning

- Result rank
  - Query refinement
Quantitative preferences

- User prefers one object to another as query result
  \[ o_1 \geq o_2 \text{ with } o_i \in O \]
Quantitative preferences

- User prefers one object to another as query result
  \[ \sim \ o_1 \geq o_2 \text{ with } o_i \in O \]

- Weighting \( w \) satisfies preference \( o_1 \geq o_2 \), if \( \text{eval}(q^\Theta, o_1, w) - \text{eval}(q^\Theta, o_2, w) \geq 0 \)
Quantitative preferences

- User prefers one object to another as query result
  \[ o_1 \geq o_2 \text{ with } o_i \in O \]

- Weighting \( w \) satisfies preference \( o_1 \geq o_2 \), if
  \[ \text{eval}(q^\Theta, o_1, w) - \text{eval}(q^\Theta, o_2, w) \geq 0 \]

- preference set \( P \) is called consistent, if transitive and reflexive closure is a partial ordered set \( \sim \) no cycles
Quantitative preferences

- User prefers one object to another as query result
  \[ \sim o_1 \geq o_2 \text{ with } o_i \in O \]

- Weighting \( w \) satisfies preference \( o_1 \geq o_2 \), if
  \( \text{eval}(q^\Theta, o_1, w) - \text{eval}(q^\Theta, o_2, w) \geq 0 \)

- Preference set \( P \) is called consistent, if transitive and reflexive closure is a partial ordered set \( \sim \) no cycles

- Assumption: \( P \) is consistent and reduced by reflexivity and transitivity
Quantitative preferences

- User prefers one object to another as query result
  \[ o_1 \geq o_2 \text{ with } o_i \in O \]

- Weighting \( w \) satisfies preference \( o_1 \geq o_2 \), if
  \[ \text{eval}(q^\Theta, o_1, w) - \text{eval}(q^\Theta, o_2, w) \geq 0 \]

- Preference set \( P \) is called consistent, if transitive and reflexive closure is a partial ordered set \( \sim \) no cycles.

- Assumption: \( P \) is consistent and reduced by reflexivity and transitivity.

- Solution: non-linear optimization problem solved by hill-climbing algorithm from Nelder/Mead.
Learning of logical combinations

3 atomic conditions: $a, b, c$

weighted DNF-formula contains exponentially many minterms
Learning of logical combinations

3 atomic conditions: \( a, b, c \)

weighted DNF-formula contains exponentially many minterms

\[
\begin{align*}
&\theta_1 \\
&\theta_2 \\
&\theta_3 \\
&\theta_4 \\
&\theta_5 \\
&\theta_6 \\
&\theta_7 \\
&\theta_8 \\
&abc \\
&\overline{abc} \\
&\overline{a}bc \\
&\overline{a}b\overline{c} \\
&\overline{ab}c \\
&\overline{abc} \\
&\overline{ab}\overline{c} \\
&\overline{a}\overline{bc} \\
&\overline{a}\overline{b}c
\end{align*}
\]

- Evaluation produces a linear formula
  \( \Rightarrow \) linear optimization problem
  \( \Rightarrow \) simplex algorithm
Learning of logical combinations

3 atomic conditions: $a, b, c$

 weighted DNF-formula contains exponentially many minterms

- Evaluation produces a linear formula
  $\implies$ linear optimization problem
  $\implies$ simplex algorithm
- discrete weight values $\implies$ MIP-problem
Learning of logical combinations

- $m$ preferences $p_i = o_{i1} \geq o_{i2}$ of a preference set $P$
Learning of logical combinations

- $m$ preferences $p_i = o_{i_1} \geq o_{i_2}$ of a preference set $P$
- $n$ orthogonal atomic conditions $c_j(o)$
Learning of logical combinations

- $m$ preferences $p_i = o_{i1} \geq o_{i2}$ of a preference set $P$
- $n$ orthogonal atomic conditions $c_j(o)$
- Logical combination as DNF-formula over potentially $2^n$ minterms $m_i$
Learning of logical combinations

- $m$ preferences $p_i = o_{i_1} \geq o_{i_2}$ of a preference set $P$
- $n$ orthogonal atomic conditions $c_j(o)$
- Logical combination as DNF-formula over potentially $2^n$ minterms $m_i$
- Every subset of the $2^n$ minterms represents an unweighted Boolean combination $\sim 2^{2n}$ variants
Learning of logical combinations

- $m$ preferences $p_i = o_{i_1} \geq o_{i_2}$ of a preference set $P$
- $n$ orthogonal atomic conditions $c_j(o)$
- Logical combination as DNF-formula over potentially $2^n$ minterms $m_i$
- Every subset of the $2^n$ minterms represents an unweighted Boolean combination $\rightsquigarrow 2^{2^n}$ variants
- Evaluation function for object $o$ and solution $l$:
  $\text{eval}(l, o) = \sum_{i=1}^{2^n} \theta_i m_i(o)$
Learning of logical combinations

- $m$ preferences $p_i = o_{i_1} \geq o_{i_2}$ of a preference set $P$
- $n$ orthogonal atomic conditions $c_j(o)$
- Logical combination as DNF-formula over potentially $2^n$ minterms $m_i$
- Every subset of the $2^n$ minterms represents an unweighted Boolean combination $\sim 2^{2^n}$ variants
- Evaluation function for object $o$ and solution $l$:
  \[ \text{eval}(l, o) = \sum_{i=1}^{2^n} \theta_i^l m_i(o) \]
- Best solution $l$:
  \[ \sum_{i=1}^{m} (\text{eval}(l, o_{i_1}) - \text{eval}(l, o_{i_2})) \implies \text{max} \]
Learning of logical combinations

- $m$ preferences $p_i = o_{i_1} \geq o_{i_2}$ of a preference set $P$
- $n$ orthogonal atomic conditions $c_j(o)$
- Logical combination as DNF-formula over potentially $2^n$ minterms $m_i$
- Every subset of the $2^n$ minterms represents an unweighted Boolean combination $\Rightarrow 2^{2n}$ variants
- Evaluation function for object $o$ and solution $l$: $\text{eval}(l, o) = \sum_{i=1}^{2^n} \theta_l^i m_i(o)$
- Best solution $l$: $\sum_{i=1}^{m} (\text{eval}(l, o_{i_1}) - \text{eval}(l, o_{i_2})) \Rightarrow \max$
- Conditions to be respected: $\text{eval}(l, o_{i_1}) - \text{eval}(l, o_{i_2}) \geq 0$ für $i = 1, \ldots, m$ and $0 \leq \theta_l^i \leq 1$ for $i = 1, \ldots, 2^n$
Learning of logical combinations

- $m$ preferences $p_i = o_{i_1} \geq o_{i_2}$ of a preference set $P$
- $n$ orthogonal atomic conditions $c_j(o)$
- Logical combination as DNF-formula over potentially $2^n$ minterms $m_i$
- Every subset of the $2^n$ minterms represents an unweighted Boolean combination $\rightsquigarrow 2^{2^n}$ variants
- Evaluation function for object $o$ and solution $l$: $\text{eval}(l, o) = \sum_{i=1}^{2^n} \theta^l_i m_i(o)$
- Best solution $l$: $\sum_{i=1}^{m} \left( \text{eval}(l, o_{i_1}) - \text{eval}(l, o_{i_2}) \right) \implies \max$
- Conditions to be respected: $\text{eval}(l, o_{i_1}) - \text{eval}(l, o_{i_2}) \geq 0$ für $i = 1, \ldots, m$ and $0 \leq \theta^l_i \leq 1$ for $i = 1, \ldots, 2^n$
- Variant: strict preferences and binary weights (MIP)
Problems

- Hard optimization problem
- Result contains $O(2^n)$ minterms, e.g. one positive conditions requires $\frac{2^n}{2}$ minterms
  $\leadsto$ merging minterms
Improvement

- $k \leq n$ many conditions may be sufficient
Improvement

- $k \leq n$ many conditions may be sufficient
- Observation: every solution with $k < n$ is also a solution on level $k + 1$ (superset of conditions)
Improvement

- $k \leq n$ many conditions may be sufficient
- Observation: every solution with $k < n$ is also a solution on level $k + 1$ (superset of conditions)
- Testing $k$ stepwise from 1 to $n$ until combination is found
Improvement

- \( k \leq n \) many conditions may be sufficient
- Observation: every solution with \( k < n \) is also a solution on level \( k + 1 \) (superset of conditions)
- Testing \( k \) stepwise from 1 to \( n \) until combination is found
- Overall costs:
  \[
  \sum_{k=1}^{n} \binom{n}{k} f(2^k, m)
  \]
  \( f(x, y) \) is cost of simplex algorithm
Improvement

- $k \leq n$ many conditions may be sufficient
- Observation: every solution with $k < n$ is also a solution on level $k + 1$ (superset of conditions)
- Testing $k$ stepwise from 1 to $n$ until combination is found
- Overall costs: $\sum_{k=1}^{n} \binom{n}{k} f(2^k, m)$
  
  $f(x, y)$ is cost of simplex algorithm
- Further improvement: start with $k > 1$
Runtime CPLEX (Minterme)

Running time in milliseconds

- runtime
  - 4000
  - 2000
  - 1000
  - 500
- number of conditions
  - 2
  - 4
  - 6
  - 8
  - 10
  - 12
  - 14
  - 16
  - 18
  - 20
- number of preferences
  - 0
  - 10000
  - 20000
  - 30000
  - 40000
  - 50000
  - 60000
  - 70000
  - 80000
Implementations

Provides for first satisfying level $k$ following alternatives

- Best result
- Simplest result
- All results (Estimating degree of freedom)
ForMaT Forschung für den Markt im Team (BMBF)

Multimediale Ähnlichkeitssuche zum Matchen, Typologisieren und Segmentieren

Partner:

- LS DBIS, Schmitt: Entwicklung einer Software, um die Suche in Multimediadaten mittels grafischer Interaktion zu verbessern
- LS Marketing und des Innovationsmanagement, Baier: Entwicklung einer Statistiksoftware zur Analyse multimedialer Daten
Softwarearchitektur

- Motivation
- CQQL
- Weighting
- Quantitative preferences
- ForMat-Projekt

- Bildkollektion
- Extraktion und Verwaltung
- Ähnlichkeitsberechnung
- Distanzen
- Similarity-Maße

- CQQL-Verknüpfung m. Gewichten
- Ähnlichkeitssuche

- SOM
- Cluster
- Klassifikation

- Präferenzen verwalten
- Bewertungen verwalten
- Gewichte lernen
- Verknüpfung lernen

- Metrikindex

- Graphische Nutzerinteraktion

- Graphisches Lern-Interface

- Extraktion und Verwaltung
- Metadaten
- Feature-Daten

- Ähnlichkeitssuche
- Feature-Daten

- Lern-Interface

- Bewertungen

- Verwaltung

- Gewichte lernen

- Metrikindex

- Bildkollektion
Bildsuche