R-Pref: Rapid Prototyping of Database Preference Queries in R

Patrick Roocks[1], Werner Kießling[1]
[1]*Institute of Computer Science, Augsburg University, D-86159 Augsburg, Germany*
{roocks,kiessling}@informatik.uni-augsburg.de

Keywords: R, Preferences, Preference SQL, Text Mining

Abstract: Preferences are a well-established framework for database queries with soft constraints. Such queries select the best objects from large data sets according to a strict partial order induced by intuitive and semantically rich preference constructors. Together with functionality like grouping and aggregation, adapted from well-known database mechanisms, a very flexible preference framework has emerged in the last decade. In this paper we present R-Pref, an implementation of the preference framework in the statistical computing language R. R-Pref comprises less than 1000 lines of code and adheres to the formal foundations of preferences. It allows rapid prototyping of new preferences and related concepts. Exemplarily we present a use case in which a simple text mining example based on pattern matching is enriched by preferences. We argue that R-Pref paves the way for rapidly exploring new fields of application for preferences. Especially new semantic constructs for preference related operations together with equivalences of preference terms, being highly important for optimization, can be quickly evaluated.

1 INTRODUCTION

Preference queries (Kießling, 2002; Chomicki, 2003) are an established concept in the database community and have been intensively studied in the last decade. Preference are an effective method to reduce very large datasets to a small set of highly interesting results and to overcome the empty result set and flooding effect. In general, a preference query selects those objects from the database that are not dominated by any other object. Therefore, preferences have shifted retrieval models from exact matching of attribute values to the notion of best matching database objects.

Preferences are strict partial orders and a set of intuitive preference constructors allows for the formulation of preference terms. According to (Stefanidis et al., 2011) Preference SQL (Kießling et al., 2011) is currently the only comprehensive approach which implements a general preference query model for databases.

In this paper we present *R-Pref* (sources and documentation at (Roocks, 2013)), an interpreter for preferences which is implemented in the *statistical computing language R* (R Core Team, 2012). We use newest language concepts like *reference classes* allowing for OOP style programming in R. The preference constructors are implemented sticking closely to their formal definition.

In R-Pref, new preference constructors can be very easily implemented and debugged. Because of this we call R-Pref a *rapid prototyping environment* for database preferences and related concepts.

A traditional example for a preference query is to find optimal products according to a consumer preference. Assume we are looking for cheap hotels close to the beach. A query searching for "minimal price and minimal distance to the beach" returns only those hotels which are not dominated in both criteria by any other hotel.

To show that R-Pref allows us to explore quite different application fields, we present a use case where preferences serve as a prefilter for a text mining application. We think that the data mining process could benefit from the introduction of *intuitive semantics* by means of preference terms, in addition to established data mining techniques. As preferences are not in the traditional scope of application for such tasks, we show the flexibility and expandability of our preference framework and its R implementation.

In our use case we will focus on text mining in a dataset of e-mails which are an example for *semi-structured data*. The mail content is unstructured, but there is a structured mail header containing sender, receiver, date and subject. Our idea is to use preferences primarily on the header columns to select the relevant mails. Afterwards, we apply simple pattern matching methods to extract the relevant information from the content. Of course, it would be also imaginable to combine more sophisticated data mining techniques with preferences which is subject to future research.

We presume that the use of preferences as a pre-filter leads to improved results as well as to a reduced parsing expense. We illustrate the principal use of preferences for searching in a mail dataset in the following example:

***Example* 1.** Consider a dataset of university internal e-mails from which we want to extract the topics a scientist is working on by searching his e-mails. Assume that the scientist Dr. Leonard Hofstadter usually sends a monthly report to his boss (Dr. Eric Gablehauser).

Therefore we primarily look out for all mails from Leonard *and* to Gablehauser. Less important to this, we pick out those which are entitled with "Monthly Report", as this is the usual subject for these reports.

To formulate this query, assume that the mail dataset of the university is stored in the table *mails* and has the columns *subject, date, from, to, content*. Consider the following Preference SQL (Kießling et al., 2011) query:

```
SELECT m, content FROM mails PREFERRING
 (    `from` IN 'hofstadter@caltech.edu'
  AND `to`   IN 'gablehauser@caltech.edu')
 PRIOR TO
  subject IN ('Monthly␣Report')
 GROUPING
   extract(month from date) AS m
```

By using **AND** we state that the preferences on the columns *from* and *to* are *equally important*. Less important to this prefer mails with a predefined subject. This wish is stated as the left hand side of **PRIOR TO**. Using the **GROUPING** construct we execute this query group-wise for every month. Thereby the aliasing with **AS** in the grouping-part is a language feature of Preference SQL (in contrast to standard SQL where this would be done in the projection).

The query retrieves the following results: For every month in which a mail with exactly these attributes exist, only this mail is returned (*exact match*). Assume that in one month Leonard sent nothing to Gablehauser, but he sent his Report to Sheldon (another scientist), and Sheldon sent it to Gablehauser, both mails entitled with "Monthly Report". According to the given preference these two mails are returned as *best matches*. Even if such mails do not exist, we retrieve all mails which are from Leonard *or* to Gablehauser, as the preference on sender/receiver is stated as a Pareto-Preference (Mails fulfilling one Pareto-condition dominate those fulfilling non of these conditions). This result might give us finally some helpful information what they are doing in this month. In Figure 1 on page 5 this preference order will be visualized.

The remainder of the paper is structured as follows: In section 2 we consider the related work regarding preferences as well as related R packages. In section 3 we introduce the specification of preferences tightly together with the implementation of R-Pref and present some examples. In section 4 we provide a use case based on information extraction from mails concerning the organization of a scientific symposium. In the final section we provide a summary and outlook.

2 RELATED WORK

The theoretical foundation of preference queries is the preference algebra which was introduced in (Kießling, 2002). In R-Pref, query statements are denoted in a very similar fashion like terms in the preference algebra. In (Stefanidis et al., 2011) a comprehensive survey of representation, composition and application of preferences is given.

The R package *sqldf* (Grothendieck, 2012) allows a manipulation of dataframes with SQL statements. Similarly to our approach established database techniques are made available to the R community. Unlike to our approach, we do not parse SQL statements but assume that the "queries" are given as nested calls of functions.

R-Pref makes use of the *igraph*-package (Csardi and Nepusz, 2006) to visualize preference orders as trees. In this package sophisticated algorithms for a neat drawing of the graphs turned out to be useful for the visualization of Better-Than-Graphs.

Additionally we use the RJDBC-package (Urbanek, 2012) which allows us to evaluate preference queries in R-Pref directly on any database system supporting JDBC. Due to the package RServe (Urbanek, 2013) R and therewith also R-Pref can be used by any Java application.

Established text mining methods (cf. (Zhang et al., 2011)) predominantly make use of statistical scoring functions like TF-IDF or LSI. In contrast to this we suggest to think about a non-numerical and more semantical approach for selecting relevant documents. Note that we merely consider the text mining approach as an idea how to combine semantics and data mining. We do not strive to compete with established data mining technologies solely with preferences.

With the package *tm* (Feinerer et al., 2008) there is a variety of text mining functions available within R.

3 PREFERENCES AND THEIR IMPLEMENTATION IN R

In this section we present the theoretical foundations of preferences according to (Kießling, 2002; Kießling, 2005) tightly together with their implementation *R-Pref.* Due to space restrictions we refer to the documentation and fully available source code on the web for further details about R-Pref (Roocks, 2013). The following code samples are restricted to the essential parts while some technical details are omitted. The code examples show that the R implementation is very near to the specification.

Definition 1 (Preference)**.** A preference $P = (A, <_P)$, where A is a set of attributes, is a strict partial order on the domain of A. Thus $<_P$ is irreflexive and transitive. Thereby $x <_P y$ is interpreted as "I like y more than x".

In R-Pref a preference is an object of the reference class `preference` having (amongst others) the fields `col` (a character-vector representing A) and a compare function `cmp` (representing $<_P$).

The result of a preference is computed by the *preference selection*, also called *winnow* by (Chomicki, 2003).

Definition 2 (Preference Selection)**.** The BMO-set of a preference $P = (A, <_P)$ on an input database relation R contains all tuples that are not dominated w.r.t. the preference. It is computed by the preference selection operator σ and finds all best matching tuples t for P, where $t.A$ is the projection to the attribute set A.

$$\sigma[P](R) := \{t \in R \mid \nexists t' \in R : t.A <_P t'.A\}$$

In the following the projection will be mostly omitted, i.e., we write just $t <_P t'$ for $t.A <_P t'.A$.

In R-Pref this is performed by the **`sigma`** function. For a preference `pref` and a dataset `tbl` the R code implementing the BMO-set definition is essentially:

```
for(i in 1:nrow(tbl))
  ind[i] = !any(pref$cmp(tbl, tbl[i,]))
res = tbl[ind,]
```

Therein **`!any`** corresponds to $\nexists$ and the call of `cmp` represents $<_P$. Of course, this is not an efficient algorithm but shows that the implementation is a close representation of its formal foundations.

3.1 Base Preference Constructors

To specify a preference, a variety of intuitive base preference constructors together with some complex preference constructors has been defined. Subsequently, we present some selected preference constructors. More preference constructors as well as their formal definition can be found in (Kießling, 2002; Kießling, 2005; Kießling et al., 2011).

Definition 3 (SCORE$_d$ Preference)**.** Assume a scoring function $f : \text{dom}(A) \rightarrow \mathbb{R}_0^+$, and some $d \in \mathbb{R}_0^+$. Then P is called a SCORE$_d$ preference, iff for $x, y \in \text{dom}(A)$:

$$x <_P y \iff f_d(x) > f_d(y)$$

where $f_d : \text{dom}(A) \rightarrow \mathbb{R}_0^+$ is defined as:

$$f_d(v) := \begin{cases} f(v) & \text{if } d = 0 \\ \left\lceil \frac{f(v)}{d} \right\rceil & \text{if } d > 0 \end{cases}$$

In R-Pref this is realized with the **`score`**`(column, scr_fnc, dval)` function in a few code lines.

An important sub-constructor of SCORE$_d$ is the BETWEEN$_d(A, [low, up])$ preference expressing the wish for a value between a *lower* and an *upper* bound. Its scoring function equals

$$f(v) = \max\{low - v, 0, v - up\}$$

In R-Pref the implementation is essentially:

```
between = function(column, low, up, ...)
  score(column, function(vals)
    pmax(low-vals, 0, vals-up), ...)
```

Thereby "..." bypasses additional arguments like the d-parameter to **`score`**. The R funtion **`pmax`** is the *parallel maximum*, which returns a vector of logicals, if *val* is a vector. Sub-constructors of BETWEEN are, e.g., the AROUND$_d(A, z)$-preference and the HIGHEST$_d(A)$-preference. We just consider their implementation as this is very close to the definition:

```
around = function(column, center, ...)
  between(column, center, center, ...)

highest = function(column, ...)
  around(column, suprema[[column]], ...)
```

Thereby `suprema` is a variable containing the maximal values of the given dataset for every numerical column, determined initially in **`sigma`**. Next to the numerical preferences there are also preferences on categorical domains, e.g., the LAYERED-preference.

Definition 4 (LAYERED$_m$ Preference)**.** Let $L = (L_1, ..., L_m)$ be an ordered list of m sets forming a partition of dom(A) for an attribute A. The preference P is a LAYERED$_m(A, (L_1, ..., L_m))$ preference if its scoring function equals

$$f(v) = i - 1 \iff x \in L_i.$$

For convenience, one of the L_i may be named "OTHERS", representing the set $\text{dom}(A) \setminus \bigcup_{j \neq i} L_j$.

The essential part in the implementation of the score-function for LAYERED is:

```
res = rep(Inf, length(vals))
  for(i in 1:length(layers))
    res[vals %in% layers[[i]]] = i-1
```

An important sub-constructor of LAYERED is the POS$(A, \textit{POS-set})$-preference which is simply implemented by:

```
pos = function(column, posset)
  layered(column, list(posset, OTHERS))
```

This assigns to all values contained in `posset` the score 0 and to all other values the score 1.

Quite similar to this, we implemented a POS_MATCHES(A, reg) preference analogous to the POS-preference, searching for a regular expression *reg* that is contained in the domain values of column *A*. Thereby we use the built-in R function **`regexec`** to process the regular expression search.

3.2 Complex Preference Constructors

In order to combine several preferences into more *complex preferences*, their relative importance has to be determined. Intuitively, people speak of "this preference is more important to me than that one" or "these preferences are all equally important to me". Equal importance is modeled by the so-called *Pareto preference* while the *Prioritization* states that one preference is more important than another preference.

To realize these complex preferences we need a notion of equality w.r.t. a preference. Therefore the SV-semantics for preferences (Kießling, 2005) have been introduced. As we will only cope with sub-constructors of score preferences in this paper, we need an equivalence relation w.r.t. a preference *P* with scoring function *f*. We state

$$x \cong_P y \Leftrightarrow f(x) = f(y)$$

which is also called *regular SV-semantics*.

Definition 5. For the *Pareto preference* $P_1 \otimes P_2$ and the *Prioritization preference* $P_1 \,\&\, P_2$, where $P_i = (A_i, <_{P_i})$, we define for all tuples $x = (x_1, x_2)$, $y = (y_1, y_2) \in \mathrm{dom}(A_1) \times \mathrm{dom}(A_2)$:

$$\begin{aligned} &(x_1, x_2) <_{P_1 \otimes P_2} (y_1, y_2) \iff \\ &\quad (x_1 <_{P_1} y_1 \wedge (x_2 <_{P_2} y_2 \vee x_2 \cong_{P_2} y_2)) \vee \\ &\quad (x_2 <_{P_2} y_2 \wedge (x_1 <_{P_1} y_1 \vee x_1 \cong_{P_1} y_1)) \end{aligned}$$

$$\begin{aligned} &(x_1, x_2) <_{P_1 \& P_2} (y_1, y_2) \iff \\ &\quad x_1 <_{P_1} y_1 \vee (x_1 \cong_{P_1} y_1 \wedge x_2 <_{P_2} y_2) \end{aligned}$$

For the equivalence relation we state for $\star \in \{\&, \otimes\}$:

$$(x_1, x_2) <_{P_1 \star P_2} (y_1, y_2) \iff x_1 \cong_{P_1} y_1 \wedge x_2 \cong_{P_2} y_2$$

In the R implementation preferences are reference classes and therefore we can overload their operators. We defined `'*.preference'` for the pareto composition and `'&.preference'` for the prioritization. We also overloaded the logical operators `&` and `|` for functions allowing for a very compact representation of the prioritization:

```
"&.preference" = function(p1, p2)
preference(col = union(p1$col, p2$col),
           cmp = p1$cmp | p1$sv & p2$cmp,
           sv  = p1$sv & p2$sv)
```

This code directly corresponds to Definition 5.

***Example* 2.** We show the R-Pref formulation for the complex preference used in Example 1:

```
p=(pos('from', 'hofstadter@caltech.edu') *
   pos('to',    'gablehauser@caltech.edu'))
 & pos('subject', 'Monthly␣Report')
```

Note that the R object `p` is again a reference class of the type "preference".

3.3 Grouped Preferences

A preference $P = (A, <_P)$ can also be evaluated in grouped mode. For a set of attributes G and a function g with domain $\mathrm{dom}(G)$ we define

$$\begin{aligned} &\sigma[P \text{ grouping } g(G)](R) := \\ &\quad \{t \in R \mid \neg \exists t' \in R : t <_P t' \wedge g(t.G) = g(t'.G)\} \end{aligned}$$

This means the BMO-set is calculated for each value of $g(\mathrm{dom}(G))$ separately and then the results are merged. The function g may be the identity but can be for example the extract(*datepart* from *date*) function if $\mathrm{dom}(G)$ is a Time&Date domain.

In R-Pref there is a **`grouping`**`(tbl, grp, pref, ...)` function realizing this functionality. Its essential code is, where `tbl` is the dataset, `grp` is $g(G)$ and `pref` is the preference:

```
do.call("rbind", lapply(split(tbl, grp),
  function(x) sigma(x, pref)))
```

In the actual implementation there is some technical overhead (ca. 70 code lines) for preparing the data structures; but the essential functionality is realized with this smart composition of built-in R functions.

***Example* 3.** Now we have everything together to encode the grouped preference selection from Example 1, where `p` is defined in Example 2:

```
res = grouping(mails,
     list(m = extract('month', date)), p)
```

Note that it is sufficient to write "`date`" in the `grp`-attribute (and not e.g., `mails$date`) because the **`grouping`** function evaluates this attribute in the scope induced by `tbl` via the built-in R functions **`substitute`** and **`eval`**. Similar techniques are used like in the R built-in function **`subset`**.

The final step is the projection to *month* and *content*. In R-Pref the function **`project`**`(tbl, lst)` realizes a projection on `tbl` where `lst` is a list of expressions to be projected wherein the grouping attribute `m` can also be referenced.

***Example* 4.** To get the same columns as in the Preference SQL query from Example 1 we finally apply the projection to the result `res` from Example 3:

```
project(res, list(m, content))
```

Note that these nested calls of **project**, **grouping**, etc. are quite near to preference relation algebra as defined in (Kießling, 2002). To define the preference p we formally write:

$$p = (\text{POS}(\text{from, 'hof...'}) \otimes \text{POS}(\text{to, 'gab...'})) \\ \& \, \text{POS}(\text{subject, 'Monthly Report'})$$

Finally the grouped preference selection together with the projection is performed by:

$$\pi_{\text{content, extract('month', date)}}(\\ \sigma[p \text{ grouping extract('month', date)}](\text{mails}) \;)$$

Despite of a different notation of the function arguments the missing aliasing, this is the same as the R-Pref commands in Examples 2–4.

Note that according to (Kießling, 2002) P grouping A for an attribute A can also expressed as a preference itself. Let $P_1 \blacklozenge P_2$ the logical and-composition (also *intersection preference*, in R-Pref the "`|`" operator) of P_1 and P_2. Assume that id_A is the identity on $\text{dom}(A)$, then we have

$$P \text{ grouping } A = \text{id}_A \blacklozenge P \; .$$

Formally, id_A is not a preference but in R-Pref we can define an **ident**`(col)` function returning a "preference" where the `cmp` field represents the identity. For example, the search for the most recent mail grouped by mail authors can be performed in R-Pref in the following ways:

```
grouping(mails, list(from), highest(date))
sigma(mails, ident(from) | highest(date))
```

Hence also such interesting interplays can be reproduced in R-Pref.

3.4 Visualization

As R is especially designed for statistical computations and data visualizations we can use such functionality to analyze characteristics of preferences. The **grouping** function offers an additional parameter `proj_agg_lst` where aggregating projections (similar to **GROUP BY** in SQL) are possible. Due to this fact it is possible to count the number of best matching mails for every month. Via the R built-in function **hist** we get a histogram visualizing the relevant mails per month. Such methods offer a quick way to determine the selectivity of preferences on a dataset.

***Example* 5.** Consider the following R code, where `p` is from Example 2:

```
res2 = grouping(mails,
   list(m = extract('month', date)), p,
   proj_agg_lst = list(n=length(m)))
hist(res2[,'n'])
```

This generates a histogram with an automatically determined bucket size showing how often BMO-sets with the same cardinality occur.

Another interesting visualization of a preference is its *Better-Than-Graph* (BTG) which is a Hasse diagram, i.e., the transitive reduction of the preference order. We use R-Pref do determine the adjacency-matrix of a given preference and we use the *igraph*-package (Csardi and Nepusz, 2006) for R to plot the graph. In Figure 1 we show an example for a partial BTG (for some correspondences) based on Example 2 from the introduction, which was created by the visualization functionality of R-Pref.

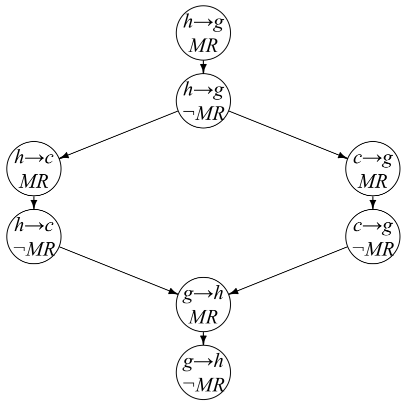

Figure 1: BTG for preference `p`

In the above figure $h \rightarrow g$ stands for mails from Hofstadter to Gablehauser, etc. ($\neg$) MR indicates if the subject of the mail equals "Monthly Report" (or not).

3.5 Other Functions of R-Pref

Up to here we could only sketch a small part of the entire R-Pref functionality; in the actual implementation (Roocks, 2013) there are more preferences constructors (e.g., EXPLICIT for used-defined orders), more parameters for the preference selection (e.g., TOP-k queries) and a plenty of SQL-like projection and aggregation functionality (e.g., complex arithmetic expressions). Also preferences on spatial domains are supported. Database joins are also readily available; the R built-in function **merge** together with a self-implemented aliasing mechanism solves this.

Due to the use of the package RJDBC (Urbanek, 2012), R-Pref comes with a direct database connection. Hence queries can not only processed on csv-datasets (the "usual" way for importing data in R) but also directly on any DBMS having a JDBC interface.

Currently, R-Pref implements the entire current Preference SQL specification, despite of algebraic optimization techniques. They are work under progress as we will describe further in the outlook in section 5.2.

4 TEXT MINING USE CASE

Assume you organize a symposium and therefore you get many mails from the participants. Therein they state their will to attend, or if they are accompanied; and they announce the titles of their talks. The latter one is what we are interested in and it is easy to extract: Mostly the participants will write something like: “My talk is about: ’...’ ”. Hence we only have to search for patterns like a colon followed by the title or for strings in quotation marks.

We took real data from the organization of our internal chair seminar. Therein 9 of 10 participants used either a Colon-[Title] or ’[Title]’ schema, hence we had an easy play with simple pattern matching. But of course this approach causes many false-positive matches. In our example we got 9 false-positive matches (where e.g., people just wrote some words in quotation marks). It is the task of an appropriate preference query to filter out the relevant mails.

As the real dataset contains confidential information, we contrived a sample dataset with eight mails from scientists to the conference organizer. Only four mails turned out to be relevant, i.e., contain actual talk titles. The typical problems therein are borrowed from our real world use case. Because of lack of space we cannot cite the entire mails here and refer to (Roocks, 2013) where the dataset (also in a pdf-version) and the R source files of this use case are available.

4.1 Iterative Preference Construction

In the following steps we will iteratively construct appropriate preferences:

***Example* 6** (First step of use case)**.** Looking at the matches for talk titles in use case we see – amongst others – the following results:

```
wolowitz | A zero-gravity human-waste
             disposal system for the ISS
wolowitz | Dr. Bernadette Rostenkowski
```

Obviously the first one is his topic while the latter one is a false positive match. How did this occur? Looking in the corresponding mail we read:

```
I forgot to say I will come together with:
Dr. Bernadette Rostenkowski
```

In contrast, the mail containing his talk started with “My topic is: ...”. This leads us to a preference for mails where “talk” or “topic” occurs in the content:

```
p1 = pos_matches('content', 'talk|topic')
```

But still some mails having ’talk’ or ’topic’ in the content produce false positive matches.

***Example* 7** (Second step of use case)**.** We also find in the matches:

```
cooper | generally understandable
cooper | The Higgs Boson as a black hole
       accelerating backwards through time
```

Of course the first one is not the topic of a talk – in this mail Sheldon Cooper makes an sarcastic comment on the organizer’s advice that all talks should be “generally understandable”. Therefore he puts this phrase in quotation marks. To filter out false matches of this kind we stipulate that titles of academic talks are usually quite long. Because of this we should prefer mails having a string with more than 30 chars in quotation marks. We put this in a prioritization chain together with our first preference from Example 6.

```
p2 = p1 &
  pos_matches('content','"[^"]{30}[^"]+"')
```

But what should we do if someone really sends us two different topics – even if we know he holds only one talk? Well, it may happen that someone changes his topic. Consider the following example.

***Example* 8** (Third step of use case)**.** We also find the following two matches:

```
hofstadter | Experimental evidences for
           the Higgs Boson as a black hole
hofstadter | Experimental observations on
            Coopers theory on Higgs Bosons
```

How could this happen? The latter mail having a later date starts with the sentence:

```
After some discussions with my colleagues
I have to change the title of my talk to:
```

This means Leonard Hofstadter changes the title of his talk (because his colleague Sheldon Cooper does not accept not to be mentioned, as he is the inventor of the theory). How can we catch this? We put a final preference in the prioritization chain: A preference taking the newest mail, realized with a HIGHEST preference on the *Date*-column:

```
p3 = p2 & highest('date')
```

This implies that within all mails being equally good according to `p2`, the newest mail is preferred, which allows the authors to revise their titles.

4.2 Preference Evaluation

So we are nearly done, but we still have not evaluated the preference. As every sender holds a talk we have to search for the best matches in every sender-group, i.e., we have to use a grouped preference where the `from` column is the grouping attribute.

***Example* 9** (Final step of use case)**.** The full preference selection for the use case is:

```
res = grouping(mails, list(from), p3)
```

But note that preferences are just soft constraints. If a mail like "I will not attend the symposium" is in the `mails` dataset it will also occur in *res*. This is no problem for the final result as there is no title-pattern in such a mail. As we aim to filter out as much as possible "senseless" information we can enrich the preference by a hard selection, requiring that "talk | topic" *has to occur* in the content.

```
subset(res,matches(content,'talk|topic'))
```

Finally applying the pattern-matching extraction methods to the remaining four mails of the dataset gives us exactly four (correct) titles of the talks. Hence we could construct an optimal prefilter for our sample dataset, just by some base preferences, a prioritization chain and finally the grouping-construct for preferences.

5 CONCLUSION

Having sketched the R-Pref system and the text mining use case we will now sum up the achievements of R-Pref and conclude with our ideas for future research in prototyping preferences and related concepts.

5.1 Summary

For the presented use case we implemented new preferences like POS_MATCHES supporting pattern matching in R. This was quite easy as we could build on the R functionality for regular expressions and the R-Pref framework. Together with a user-friendly R-IDE (we used "RStudio") R-Pref turns out to be a comfortable rapid-prototyping environment allowing to experiment with different preferences and related approaches. New constructors can be implemented in a few minutes and in few lines of code. The easily applicable internal visualization functionality of R (histograms, bar plots, etc.) and external packages like *igraph* can be used to visualize the results and characteristics of preferences. Additionally preferences and statistical approaches can be compared as R is especially designed for statistic calculations.

Even with R being originally designed for statistical applications, nowadays its application scope is much more widespread to due a plenty of packages for e.g., databases or data mining. New developments like reference classes together with operator overloading offer the possibility for an extremely concise coding, which we use for the composition of preferences in an algebraic style. Due to all these considerations, it was the logical consequence to make our comprehensive preference framework available for R.

In analogy to the algebraic optimization rules of Preference SQL like "Push preference over join" one can see our text mining use case under the paradigm *"Push preference into code"*. Therein "code" represents the common text mining techniques as e.g., clustering, summarizing or finding associations. For future research, established concepts for exploring semi-structured data can be combined with the approach of preferences.

5.2 Outlook

The development of R-Pref just started in November 2012 and this project is still in the beginning. Although we are supporting a comprehensive preference framework, there is still a lot of work to do.

In one part of our project we are developing an R-based automatic correctness test application for the Preference SQL system (Kießling et al., 2011). Therein datasets and queries are randomly generated and executed on both systems, R-Pref and Preference SQL. Afterwards the results are compared and differences are returned as potential errors. Of course, such an approach just offers "probable correctness" but it is highly improbable that both implementations have exactly the same errors. The specification-near coding style of R-Pref together with a sufficient large search space (of queries and datasets) gives a strong hint for correctness in general.

In (Hafenrichter and Kießling, 2005) sophisticated optimization techniques like *"Push preference over join"* are introduced. In the context of R these can be considered as transformations on expressions. Syntactically, due to `expression([query])` R offers a neat semantic layer to manipulate function calls before evaluating. Because R-Pref queries are near to the relational algebraic representation it seams reasonable to study different optimization techniques based on transformations of R expressions. We are working on an optimizer in a "formal style" just consisting of a set of algebraic optimization rules and their preconditions. Therein the optimization rules should not be subjected to an error-prone application-specific parsing process, but highly benefit from the general semantic structure of expressions and references classes in R.

Regarding the text mining use case, at the current stage of development this project cannot compete with established data mining techniques. But we think that the data mining process will benefit from the semantical structure of preferences and the *best matches only* query model in many aspects. Most data mining algorithms are based on many weighting factors, which have little intuitive meaning to the data analyst. In contrast, preferences terms are quite intuitively understandable and therefore we think the field of semantics in query languages is an interesting research field for decision support, data mining, etc.; a field where the statistical computing language R is very popular. A close connection of preferences and established algorithms in this area might lead to substantial new results.

In a nutshell, our vision is to use R-Pref as an experimental incubator for rapidly exploring new research ideas. Ideas found promising will then be implemented efficiently in our main Preference SQL system.

ACKNOWLEDGEMENT

This work has been funded by the Bavarian Ministry of Economic Affairs, Infrastructure, Transport and Technology, grant no. IUK-1109-0003//IUK398/002.

REFERENCES

Chomicki, J. (2003). Preference Formulas in Relational Queries. In *TODS '03: ACM Transactions on Database Systems*, volume 28, pages 427–466, New York, NY, USA. ACM Press.

Csardi, G. and Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal*, Complex Systems:1695.

Feinerer, I., Hornik, K., and Meyer, D. (2008). Text Mining Infrastructure in R. *Journal of Statistical Software*, 25(5):1–54.

Grothendieck, G. (2012). *sqldf: Perform SQL Selects on R Data Frames*. R package version 0.4-6.4.

Hafenrichter, B. and Kießling, W. (2005). Optimization of Relational Preference Queries. In *Proceedings of the 16th Australasian database conference - Volume 39*, ADC '05, pages 175–184, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Kießling, W. (2002). Foundations of Preferences in Database Systems. In *VLDB '02: Proceedings of the 28th International Conference on Very Large Data Bases*, pages 311–322, Hong Kong, China. VLDB.

Kießling, W. (2005). Preference Queries with SV-Semantics. In Haritsa, J. R. and Vijayaraman, T. M., editors, *COMAD '05: Advances in Data Management 2005, Proceedings of the 11th International Conference on Management of Data*, pages 15–26, Goa, India. Computer Society of India.

Kießling, W., Endres, M., and Wenzel, F. (2011). The Preference SQL System - An Overview. *Bulletin of the Technical Commitee on Data Engineering, IEEE Computer Society*, 34(2):11–18.

R Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Roocks, P. (2013). R-Pref Documentation, Sources and use case `http://ursaminor.informatik.uni-augsburg.de/trac/wiki/R-Pref`.

Stefanidis, K., Koutrika, G., and Pitoura, E. (2011). A Survey on Representation, Composition and Application of Preferences in Database Systems. *ACM Transaction on Database Systems*, 36(4).

Urbanek, S. (2012). *RJDBC: Provides access to databases through the JDBC interface*. R package version 0.2-1.

Urbanek, S. (2013). *Rserve: Binary R server*. R package version 0.6-8.1.

Zhang, W., Yoshida, T., and Tang, X. (2011). A comparative study of TF*IDF, LSI and multi-words for text classification. *Expert Systems with Applications*, 38(3):2758 – 2765.