

Good Relations with R

Kurt Hornik David Meyer

Motivation

Meyer, Leisch & Hornik (2003), “The Support Vector Machine under test”, *Neurocomputing*: Large scale benchmark analysis of performance of SVMs for classification and regression problems. Lead to:

Hothorn, Leisch, Zeileis & Hornik (2005), “The design and analysis of benchmark experiments”, *Journal of Computational and Graphical Statistics*.

Hornik & Meyer (2007), “Deriving consensus rankings from benchmarking experiments”, Proceedings of GfKI 2006.

In particular: how can the results on individual data sets be aggregated?
More generally: how can possibly partial preference relations be aggregated?

Such issues are dealt with in social choice (going back to Borda and Condorcet), group choice, multi criteria decision making, . . .

Consensus relations

Aggregation of individual relations amounts to determining so-called *consensus relations*, e.g., as a *central relation* R which minimizes

$$\Phi(R) = \sum_{b=1}^B w_b d(R_b, R)$$

for a suitable dissimilarity measure d over a suitable class of relations R (e.g., preferences or linear orders).

Applications abound: rank proposals, candidates, journals, web pages, . . . , based on possibly incomplete individual rankings.

Relations

Given k sets of objects X_1, \dots, X_k , a k -ary relation R on $D(R) = (X_1, \dots, X_k)$ is a subset $G(R)$ of the Cartesian product $X_1 \times \dots \times X_k$. I.e.,

- $D(R)$, the *domain* of R , is a k -tuple of sets
- $G(R)$, the *graph* of R , is a set of k -tuples

To provide a faithful computational model, need tuples (where R vectors can serve reasonably well) and sets.

Sets in base R

A set is a collection of distinct objects.

Base R provides some functionality for set computations (`union`, `intersect`, `setdiff`, ...), but no data structures, and e.g.

```
R> union(list(1), list("1"))
```

```
[[1]]
```

```
[1] 1
```

```
[[2]]
```

```
[1] "1"
```

```
R> intersect(list(1), list("1"))
```

```
[[1]]
```

```
[1] "1"
```

(Part of the “problem” is that `match` is used for comparing elements.)

Package sets

Package **sets** provides data structures and basic operations for ordinary sets, and generalizations such as fuzzy sets, multisets, and fuzzy multisets (and tuples).

Sets can be created via `set` or `as.set`.

Operations include union, intersection, Cartesian product, etc., mostly also available as binary operators (`|`, `&`, `*`, etc.).

```
R> A <- set(1)
R> B <- set("1")
R> A | B
{1, 1}
R> A & B
{}
```

Printing by default does not quote character strings; comparison is performed via `identical`.

Power sets and outer products

Power sets can be obtained via 2^S .

Using `set_outer`, one can apply a function on all factorial combinations of the elements of two sets.

```
R> S <- set(1, 2, 3)
```

```
R> PS <- 2^S
```

```
R> set_outer(PS, PS, FUN = set_is_subset)
```

	{}	{1}	{2}	{3}	{1, 2}	{1, 3}	{2, 3}	{1, 2, 3}
{}	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
{1}	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
{2}	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
{3}	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
{1, 2}	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE
{1, 3}	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE
{2, 3}	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE
{1, 2, 3}	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

Food for thought

Sets are really tricky.

Their elements should be “distinct”, but how should they be compared? (Using `==`, `all.equal`, `identical`, ... ?)

Elements of sets have no position: hence, positional subscripting is disallowed. Iteration is used for accessing the elements, currently (rather low-level) via `lapply/as.list`.

Work on a general iteration mechanism for (base) R is under way.

Fuzzy sets

Fuzzy sets are sets whose elements have degrees of membership. Introduced by Zadeh (1965) as an extension of the classical notion of a set, extending the basic set operations \cap , \cup , \neg to the \min , \max , $1 -$ of the corresponding membership values.

Modern fuzzy set theory knows a variety of other extension (“fuzzy logics”) via t-norms, t-conorms, and negations.

Package **sets** supports the most popular fuzzy logic families (drastic, product, Lukasiewicz, Fodor, Frank, Hamacher, . . .).

Package relations

Package **relations** provides data structures and algorithms for k -ary relations with arbitrary domains, featuring relational algebra, predicate functions, and fitters for consensus relations.

Relations can be created via `relation` by giving the graph, characteristic function or incidences and possibly the domain, or via `as.relation` (e.g., unordered factors coerced to equivalence relations; ordered factors and numeric vectors to order relations, data frames taken as relation tables).

Characteristic function: membership function of the graph.

Incidences: array of memberships of the corresponding tuples in the graph.

Under the hood

The R universe features many “relational” data structures (cluster partitions correspond to equivalence relations; graphs, hypergraphs and networks; ...).

Relations are implemented as an S3 class which allows for a variety of internal representations (even though currently, only dense array representations of the incidences are employed). (“Containers”.)

Computations on relations are based on high-level generic getters for the basic constituents: `relation_domain`, `relation_graph`, `relation_charfun`, `relation_incidence`.

Example

```
R> R <- as.relation(c(1, 2))
```

```
R> relation_domain(R)
```

Relation domain:

A pair with elements:

```
{1, 2}
```

```
{1, 2}
```

```
R> relation_graph(R)
```

Relation graph:

A set with pairs:

```
(1, 1)
```

```
(1, 2)
```

```
(2, 2)
```

```
R> relation_incidence(R)
```

Incidences:

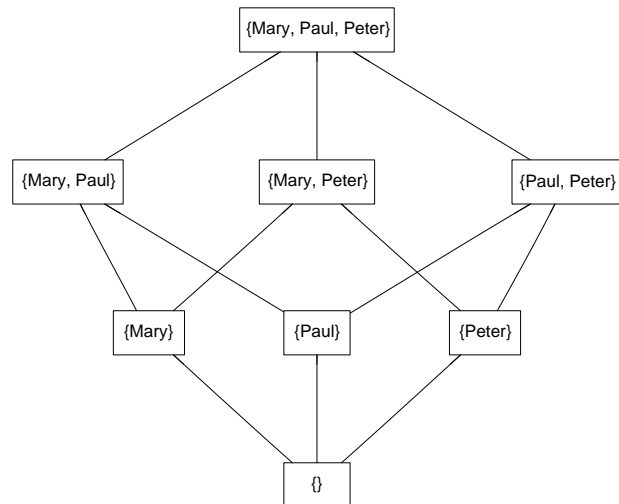
```
1 2
```

```
1 1 1
```

```
2 0 1
```

Example

```
R> S <- set("Peter", "Paul", "Mary")  
R> R <- relation(incidence = set_outer(2^S, '<='))  
R> R  
A binary relation of size 8 x 8.  
R> plot(R)
```



Endorelations and predicates

Endorelations are binary relations with domain $D = (X, X)$.

Such relations can be reflexive, symmetric, transitive,

Important combinations of the basic properties include

equivalence reflexive, symmetric, and transitive

preference complete, reflexive, and transitive (also known as “weak order”)

linear order antisymmetric preference

These properties can be tested for using `relation_is_foo` predicates.

The `summary` method for relations applies all available predicates.

Basic operations

Rich collection of basic operations, including

- Complement and dual
- Comparisons (using the natural ordering), meet and join
- Composition, union, intersection, difference
- Projection, product and various joins
- Transitive reduction and closure
- Plotting (via **Rgraphviz**) for certain endorelations (using Hasse diagrams)

Implements relational algebra of Codd (1970) using convenient binary operators.

Ensembles

Relation ensembles are collections of relations $R_b = (D_b, G_b)$, $1 \leq b \leq B$ with identical domains, i.e., $D_1 = \dots = D_B$.

Implemented as suitably classed lists of relation objects, making it possible to use `lapply` for computations on the individual relations in the ensemble.

Available methods for relation ensembles include those for subscripting, `c`, `t`, `rep`, and `print`.

Dissimilarities

Several methods for computing dissimilarities between (ensembles of) relations, with default the symmetric difference distance (the cardinality of the symmetric difference of two relations, i.e., the number of tuples contained in exactly one of two relations).

Characterizable as the least element moves distance in the lattice of relations on the same domain under the natural (set inclusion of the graphs) order. For preference relations: Kemeny-Snell distance.

In addition, Cook-Kress and Cook-Kress-Seiford distances.

Allows for dissimilarity based analysis of relation ensembles (clustering, scaling, ...).

Consensus relations

Several methods for obtaining consensus relations, including Borda, Condorcet and Copeland methods, but most importantly for finding central relations minimizing weighted average symmetric distance

$$\Phi(R) = \sum_{b=1}^B w_b d(R_b, R)$$

over suitable “families” of relation (e.g., equivalence, preferences and linear orders).

Accomplished by reformulating the consensus problem as a binary linear program

$$\sum_{i,j} c_{ij}(w_1, \dots, w_B, R_1, \dots, R_B) x_{ij} \rightarrow \max$$

for the 0/1 incidences x_{ij} of the consensus relation.

Consensus relations

Allows using solvers from packages **Rcplex**, **Rglpk**, **Rsymphony** and **lpSolve**.

(Encapsulates creation and solution of MILPs, to be spun off into an optimization infrastructure package eventually.)

Always possible to find *all* solutions via poor person's branch and cut (only `lp_solve` provides some solver support for this).

For equivalences and preferences, can specify the desired number of equivalence classes. For this, consensus problem is reformulated as a binary quadratic program, currently solved via linearization, with a direct BQP/**Rcplex** method under way.

Example: SVM Benchmarking

Results for benchmarking 17 classification methods on 21 data sets: relation ensemble of length 21 with encoding

$$I(R_b)_{i,j} = \begin{cases} 1 & \text{if method } i \text{ did not significantly outperform method } j \\ & \text{on data set } b \text{ (was } \leq) \\ 0 & \text{otherwise} \end{cases}$$

Load the data set:

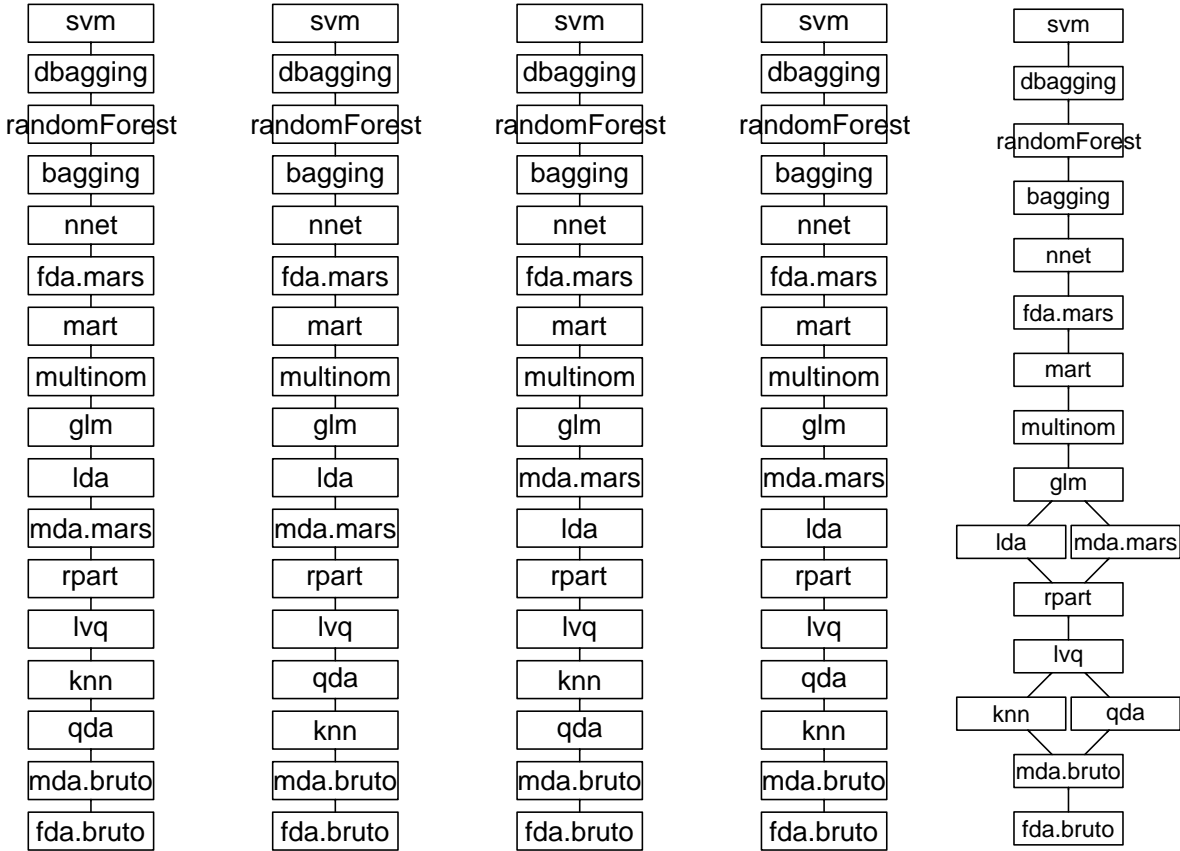
```
R> data("SVM_Benchmarking_Classification")
R> SVM_Benchmarking_Classification
An ensemble of 21 relations of size 17 x 17.
```

Fit all consensus linear orders and preferences:

```
R> cens_L <- relation_consensus(SVM_Benchmarking_Classification,
+   "SD/L", all = TRUE)
R> cens_P <- relation_consensus(SVM_Benchmarking_Classification,
+   "SD/P", all = TRUE)
```

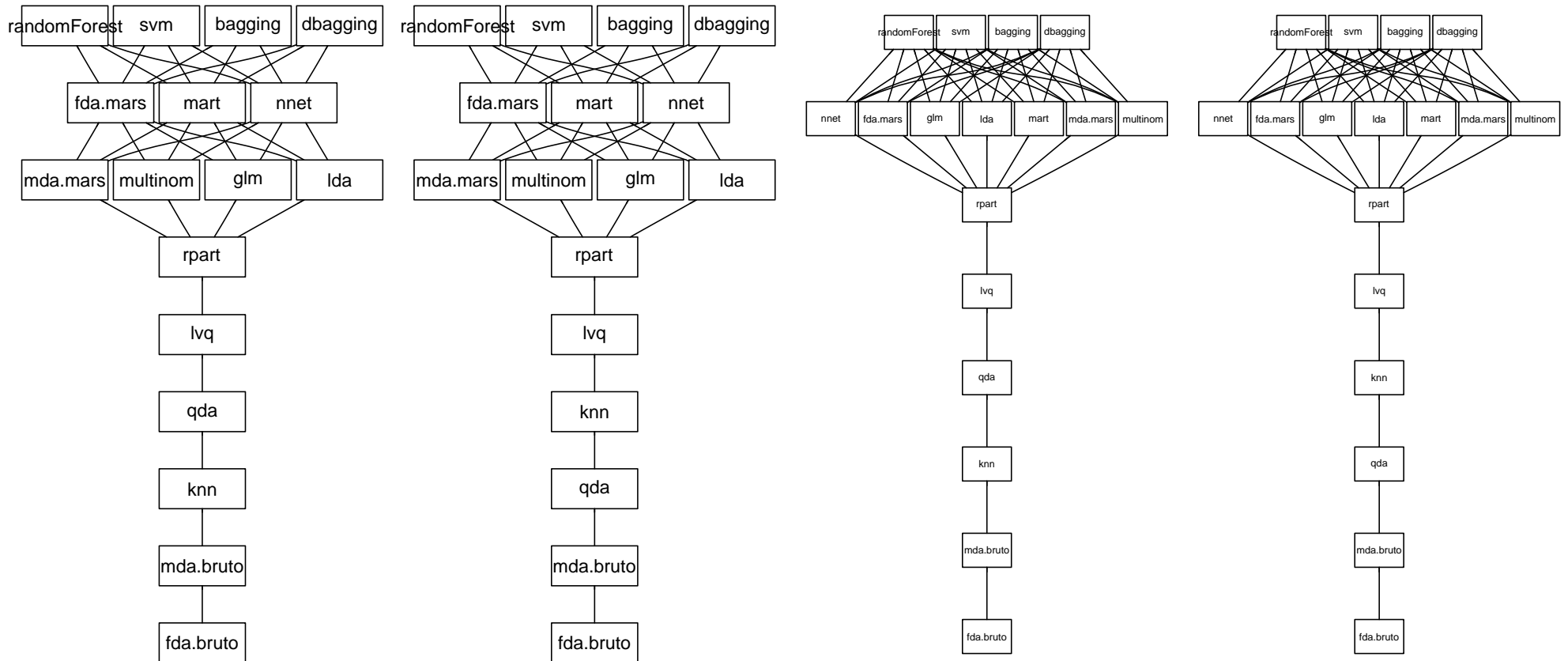
Consensus Relations: Linear Orders

```
R> plot(c(cens_L, min(cens_L)), layout = c(1, 5))
```



Consensus Relations: Preferences

```
R> plot(cens_P, layout = c(1, 4))
```



Extensions

Of course, we can also do ...

- Fuzzy relations
- Prototype-based partitioning (“clustering”) of relation ensembles
- Social choice (e.g., determine the/all “ k -winners”):

```
R> relation_choice(SVM_Benchmarking_Classification, k = 4)  
{bagging, dbagging, randomForest, svm}
```
- and much more.

Coordinates

Kurt Hornik, David Meyer
Wirtschaftsuniversität Wien
Augasse 2–6, A-1090 Wien

E-mail: *Firstname.Lastname@wu-wien.ac.at*

WWW: <http://statmath.wu-wien.ac.at/~hornik/>
<http://wi.wu-wien.ac.at/~meyer/>