Classes for record linkage of big data sets

Andreas Borg, Murat Sariyar

May 10, 2011

As of version 0.3, the package RecordLinkage includes extensions to overcome the problem of high memory consumption that arises when processing a large number of records (i.e. building record pairs out of ≥ 1000 records without blocking). This is achieved by blockwise creation of comparison patterns instead of computing and storing the whole set of patterns at once, which was the only choice in the former version. In addition, an embedded SQLite database is used through package RSQLite to perform blocking, application of phonetic codes or string metrics and creation of comparison patterns. This allows to make use of the efficient data structures (e.g. indexing) implemented in the SQLite engine.

In order to facilitate a tidier design, S4 classes and methods were used to implement the extensions. In favor of backward compatibility and development time, plans of a complete transition to S4 were dismissed. Nevertheless, the existing functions were joined with their new counterparts, resulting in methods which dispatch on the new S4 as well as on the existing S3 classes. This approach combines two advantages: First, existing code using the package still works, second, the new classes and methods offer (nearly) the same interface, i.e. the necessary function calls for a linkage task differ only slightly. An exception is `getPairs`, whose arguments differ from the exisiting version (see man page).

1 Defining data and comparison parameters

The existing S3 class `"RecLinkData"` is supplemented by the S4 classes `"RLBigDataLinkage"` and `"RLBigDataDedup"` for linking two datasets and deduplication of one dataset respectively. Both share the common abstract superclass `"RLBigData"`.

```
> library(RecordLinkage)

data.table 1.6
Quick start guide : vignette("datatable-intro")
Homepage : http://datatable.r-forge.r-project.org/
Help : help("data.table") or ?data.table (includes fast start examples)
RecordLinkage library
[c] IMBEI Mainz

> showClass("RLBigData")

Virtual Class "RLBigData" [package "RecordLinkage"]
```

```
Slots:

Name:    frequencies      blockFld    excludeFld
Class:       numeric          list       numeric

Name:      strcmpFld     strcmpFun   phoneticFld
Class:       numeric     character       numeric

Name:    phoneticFun           drv           con
Class:     character     DBIDriver DBIConnection

Name:         dbFile
Class:     character

Known Subclasses: "RLBigDataDedup", "RLBigDataLinkage"
```

> showClass("RLBigDataDedup")

```
Class "RLBigDataDedup" [package "RecordLinkage"]

Slots:

Name:           data      identity   frequencies
Class:    data.frame        factor       numeric

Name:       blockFld    excludeFld     strcmpFld
Class:          list       numeric       numeric

Name:      strcmpFun   phoneticFld   phoneticFun
Class:     character       numeric     character

Name:            drv           con        dbFile
Class:     DBIDriver DBIConnection     character

Extends: "RLBigData"
```

> showClass("RLBigDataLinkage")

```
Class "RLBigDataLinkage" [package "RecordLinkage"]

Slots:

Name:          data1         data2     identity1
Class:    data.frame    data.frame        factor

Name:      identity2   frequencies      blockFld
Class:        factor       numeric          list

Name:     excludeFld     strcmpFld     strcmpFun
Class:       numeric       numeric     character
```

```
Name:    phoneticFld   phoneticFun           drv
Class:       numeric     character     DBIDriver

Name:            con        dbFile
Class: DBIConnection     character

Extends: "RLBigData"
```

For the two non-virtual classes, the constructor-like function `RLBigDataDedup` and `RLBigDataLinkage` exist, which correspond to `compare.dedup` and `compare.linkage` for the S3 classes and share most of their arguments. In contrast to the latter, these functions do not create the whole set of comparison patterns but only instantiate an object that holds all the information necessary to construct these pairs on demand.

The following example shows the basic usage of the constructors, for details consult their documentation.

```
> data(RLdata500)
> data(RLdata10000)
> rpairs1 <- RLBigDataDedup(RLdata500, identity = identity.RLdata500,
+     blockfld = list(1, 3), strcmp = 1:4)
> s1 <- 471:500
> s2 <- sample(1:10000, 300)
> identity2 <- c(identity.RLdata500[s1],
+     rep(NaN, length(s2)))
> dataset <- rbind(RLdata500[s1, ], RLdata10000[s2,
+     ])
> rpairs2 <- RLBigDataLinkage(RLdata500,
+     dataset, identity1 = identity.RLdata500,
+     identity2 = identity2, phonetic = 1:4,
+     exclude = "lname_c2")
```

In order to create comparison patterns, the following backend functions exist, which are usually not directly executed by the user:

begin Constructs an SQL statement to execute blocking, phonetic code, string comparison and building comparison patterns and sends this query to the underlying SQLite database. Takes as argument the object to process.

nextPairs Fetches a block of patterns after the query has been send. Takes as arguments the object from which to fetch and the maximum number of comparison patterns to return.

clear Clears the result set after comparison patterns have been fetched. Takes as argument the object to process.

```
> rpairs1 <- begin(rpairs1)
> nextPairs(rpairs1, 10)

   id1 id2  fname_c1 fname_c2 lname_c1 lname_c2
1    1   8 0.0000000       NA 1.000000       NA
2    1  64 0.6190476       NA 1.000000       NA
```

```
3    1 141 0.5619048       NA 1.000000       NA
4    1 185 0.6190476       NA 1.000000       NA
5    1 217 0.6761905       NA 1.000000       NA
6    1 248 0.6761905       NA 1.000000       NA
7    1 268 0.6011905       NA 1.000000       NA
8    1 325 0.3952381       NA 1.000000       NA
9    1 428 0.5396825       NA 1.000000       NA
10   1 174 1.0000000       NA 0.447619       NA
   by bm bd is_match
1   0  0  0        0
2   0  1  0        0
3   0  0  0        0
4   0  0  0        0
5   0  0  0        0
6   0  0  0        0
7   0  0  0        0
8   0  0  0        0
9   0  0  0        0
10  0  0  0        0
```

```
> clear(rpairs1)
```

```
[1] TRUE
```

2 Supervised classification

The existing function `classifySupv` was transformed to a S4 method which handles the old S3 object (`"RecLinkData"`) as well as the new classes. However, at the moment a classificator can only be trained with an object of class `"RecLinkData"`.

```
> train <- getMinimalTrain(compare.dedup(RLdata500,
+     identity = identity.RLdata500, blockfld = list(1,
+         3)))
> rpairs1 <- RLBigDataDedup(RLdata500, identity = identity.RLdata500)
> classif <- trainSupv(train, "rpart", minsplit = 2)
> result <- classifySupv(classif, rpairs1)
```

The result is an object of class `"RLResult"` which contains the indices of links and optionally possible links.

```
> showClass("RLResult")
```

```
Class "RLResult" [package "RecordLinkage"]

Slots:

Name:           data         links possibleLinks
Class:     RLBigData        matrix        matrix

Name:         nPairs
Class:       numeric
```

A contingency table can be viewed via `getTable`, various error measures are calculated by `getErrorMeasures`.

```
> getTable(result)
```

```
           classification
true status      N      P      L
      FALSE 124689      0     11
      TRUE       0      0     50
```

```
> getErrorMeasures(result)
```

```
$alpha
[1] 0

$beta
[1] 8.821171e-05

$accuracy
[1] 0.9999118

$precision
[1] 0.8196721

$sensitivity
[1] 1

$specificity
[1] 0.9999118

$ppv
[1] 0.8196721

$npv
[1] 1
```

3 Weight-based classification

As with `"RecLinkData"` objects, weight-based classification with `"RLBigData*"` classes includes weight calculation and classification based on one or two thresholds, dividing links, non-links and, if desired, possible links. The following example applies classification with Epilink (see documentation of `epiWeights` for details):

```
> rpairs1 <- epiWeights(rpairs1)
> result <- epiClassify(rpairs1, 0.5)
> getTable(result)
```

```
           classification
true status      N      P      L
      FALSE 124699      0      1
      TRUE       4      0     46
```

By default, the weights for each individual record pair are stored in the associated database, which speeds up subsequent classification significantly. If the resulting disk usage is an issue, this behaviuor can be changed as follows:

- In the case of weight calculation with an EM algorithm by calling `emWeights` with argument `save.weights = FALSE`. This results in only $2^{\#\text{attributes}}$ per-pattern weights being stored.
- In the case of Epilink weights, `epiWeights` can be called directly. In this case, weights are calculated during classification but are not saved in memory.

4 Evaluation and results

In addition to `getTable` and `getErrorMeasures`, `getPairs`, which was redesigned as a versatile S4 method, is an important tool to inspect data and linkage results. For example, the following code extracts all links with weights greater or equal than 0.7 from the result set obtained in the last example:

```
> getPairs(result, min.weight = 0.7, filter.link = "link")

   id fname_c1 fname_c2 lname_c1 lname_c2   by bm
1 290    HELGA ELFRIEDE   BERGER     <NA> 1989  1
2 466    HELGA ELFRIEDE   BERGER     <NA> 1989  1
3
4 467   ULRIKE   NICOLE   BECKRR     <NA> 1982  8
5 472   ULRIKE   NICOLE   BECKER     <NA> 1982  8
6
7 313   URSULA   BIRGIT  MUELLRR     <NA> 1940  6
8 457   URSULA   BIRGIT  MUELLER     <NA> 1940  6
9
  bd is_match Class    Weight
1 18
2 28     TRUE     L 0.7786012
3
4  4
5  4     TRUE     L 0.7293529
6
7 15
8 15     TRUE     L 0.7293529
9
```

A frequent use case is to inspect misclassifed record pairs; for this purpose two shortcuts are included that call `getPairs` with appropriate arguments:

```
> getFalsePos(result)

   id fname_c1 fname_c2 lname_c1 lname_c2   by bm
1 388   ANDREA     <NA>    WEBER     <NA> 1945  5
2 408   ANDREA     <NA>  SCHMIDT     <NA> 1945  2
3
```

```
  bd is_match Class    Weight
1 20
2 20    FALSE     L 0.5067013
3
```

```
> getFalseNeg(result)
```

```
    id fname_c1 fname_c2 lname_c1 lname_c2   by
1  353     INGE     <NA>   SEIDEL     <NA> 1949
2  355    INGEU     <NA>   SEIDEL     <NA> 1949
3
4  285    ERIKA     <NA>    WEBER     <NA> 1995
5  379    ERIKA     <NA>    WEBER     <NA> 1992
6
7  127     KARL     <NA>    KLEIN     <NA> 2002
8  142     KARL     <NA>   KLEIBN     <NA> 2002
9
10  37 HARTMHUT     <NA> HOFFMSNN     <NA> 1929
11  72  HARTMUT     <NA> HOFFMANN     <NA> 1929
12
   bm bd is_match Class    Weight
1   9  4
2   8  4     TRUE     N 0.4948059
3
4   2  1
5   2 29     TRUE     N 0.4782410
6
7   6 20
8   6 29     TRUE     N 0.4692532
9
10 12 29
11 12 29     TRUE     N 0.4081096
12
```