The RecordLinkage Package: Detecting Errors in Data

by Murat Sariyar and Andreas Borg

Abstract Record linkage deals with detecting homonyms and mainly synonyms in data. The package **RecordLinkage** provides means to perform and evaluate different record linkage methods. A stochastic framework is implemented which calculates weights through an EM algorithm. The determination of the necessary thresholds in this model can be achieved by tools of extreme value theory. Furthermore, machine learning methods are utilized, including decision trees (**rpart**), bootstrap aggregating (**bagging**), ada boost (**ada**), neural nets (**nnet**) and support vector machines (**svm**). The generation of record pairs and comparison patterns from single data items are provided as well. Comparison patterns can be chosen to be binary or based on some string metrics. In order to reduce computation time and memory usage, blocking can be used. Future development will concentrate on additional and refined methods, performance improvements and input/output facilities needed for real-world application.

Introduction

When dealing with data from different sources that stem from and represent one realm, it is likely that homonym and especially synonym errors occur. In order to reduce these errors either different data files are linked or one data file is deduplicated. Record linkage is the task of reducing such errors through a vast number of different methods. These methods can be divided into two classes. One class consists of stochastic methods based on the framework of Fellegi and Sunter (1969). The other class comprises non-stochastic methods from the machine learning context. Methods from both classes need preliminary steps in order to generate data pairs from single data items. These record pairs are then transformed into comparison patterns. An exemplary comparison pattern is of the form $\gamma = (1,0,1,0,1,0,0,0)$ where only agreement and non-agreement of eight attributes are evaluated. This transformation and other preprocessing steps are described in the next section. Stochastic record linkage is primarily defined by the assumption of a probability model concerning probabilities of agreement of attributes conditional on the matching status of the underlying data pair. Machine learning methods reduce the problem of record linkage to a classification problem.

The package **RecordLinkage** is designed to facilitate the application of record linkage in R. The idea for this package evolved whilst using R for record linkage of data stemming from a German cancer registry. An evaluation of different methods thereafter lead to a large number of functions and data structures. The organisation of these functions and data structures as an R package eases the evaluation of record linkage methods and facilitates the application of record linkage to different data sets. These are the main goals behind the package described in this paper.

RecordLinkage is available from our project home page on R-Forge[1] as well as from CRAN.

Data preprocessing

First steps in data preprocessing usually include standardization of data, for example conversion of accented characters or enforcing a well-defined date format. However, such methods are not included in **RecordLinkage**. In the package, the user is responsible for providing the data in the form the package expects it.

Data must reside in a data frame where each row holds one record and columns represent attributes. The package includes two example data sets, `RLdata500` and `RLdata10000`, which differ in the number of records.[2] The following example shows the structure of `RLdata500`.

```
> library(RecordLinkage)
> data(RLdata500)
> RLdata500[1:5, ]

  fname_c1 fname_c2 lname_c1 lname_c2   by bm bd
1  CARSTEN     <NA>    MEIER     <NA> 1949  7 22
2     GERD     <NA>    BAUER     <NA> 1968  7 27
3   ROBERT     <NA> HARTMANN     <NA> 1930  4 30
4   STEFAN     <NA>    WOLFF     <NA> 1957  9  2
5     RALF     <NA>  KRUEGER     <NA> 1966  1 13
```

The fields in this data set are first name and family name, each split into a first and second component, and the date of birth, with separate components for day, month and year.

Column names are reused for comparison patterns (see below). If a record identifier other than the row number is desired, it should be included as a separate column instead of using `row.names`.

[1] http://r-forge.r-project.org/projects/recordlinkage/

[2] The data were created randomly from German name statistics and have no relation to existing persons.

Building comparison patterns

We include two functions for the creation of comparison patterns from data sets: `compare.dedup` for deduplication of a single data set and `compare.linkage` for linking two data sets together. In the case of three or more data sets, iteratively two of them are linked and replaced by the data set which is the result of the linkage. This leads to $n-1$ linkages for n data sets.

Both compare functions return an object of class `"RecLinkData"` which includes, among other components, the resulting comparison patterns as component `pairs`. In the following, such an object will be referred to as a *data object*.

```
> rpairs <- compare.dedup(RLdata500,
+   identity = identity.RLdata500)
> rpairs$pairs[1:5, ]

  id1 id2 fname_c1 fname_c2 lname_c1 lname_c2 by
1   1   2        0       NA        0       NA  0
2   1   3        0       NA        0       NA  0
3   1   4        0       NA        0       NA  0
4   1   5        0       NA        0       NA  0
5   1   6        0       NA        0       NA  0
  bm bd is_match
1  1  0        0
2  0  0        0
3  0  0        0
4  0  0        0
5  1  0        0
```

The printed slice of `datapairs$pairs` shows the structure of comparison patterns: The row numbers of the underlying records are followed by their corresponding comparison vector. Note that a missing value in either of two records results in a `NA` in the corresponding column of the comparison pattern. The classification procedures described in the following sections treat these `NA`s as zeros by default. If this behaviour is undesired, further preprocessing by the user is necessary. Column `is_match` denotes the true matching status of a pair (0 means non-match, 1 means match). It can be set externally by using the optional argument `identity` of the compare functions.[3] This allows evaluation of record linkage procedures based on labeled data as a gold standard.

Blocking

Blocking is the reduction of the amount of data pairs through focusing on specified agreement patterns.

Unrestricted comparison yields comparison patterns for all possible data pairs: $n(n-1)/2$ for deduplication of n records, $n \cdot m$ for linking two data sets with n and m records. Blocking is a common strategy to reduce computation time and memory consumption by only comparing records with equal values for a subset of attributes, called blocking fields. A blocking specification can be supplied to the `compare` functions via the argument `blockfld`. The most simple specification is a vector of column indices denoting the attributes on which two records must agree (possibly after applying a phonetic code, see below) to appear in the output. Combining several such specifications in a list leads to the union of the sets obtained by the individual application of the specifications. In the following example, two records must agree in either the first component of the first name or the complete date of birth to appear in the resulting set of comparison patterns.

```
> rpairs <- compare.dedup(RLdata500,
+   blockfld = list(1, 5:7),
+   identity = identity.RLdata500)
> rpairs$pairs[c(1:3, 1203:1204), ]

     id1 id2 fname_c1 fname_c2 lname_c1 lname_c2
1     17 119        1       NA        0       NA
2     61 106        1       NA        0       NA
3     61 175        1       NA        0       NA
1203  37  72        0       NA        0       NA
1204  44 339        0       NA        0       NA
     by bm bd is_match
1     0  0  0        0
2     0  0  1        0
3     0  0  1        0
1203  1  1  1        1
1204  1  1  1        0
```

Phonetic functions and string comparators

Phonetic functions and string comparators are similar, yet distinct approaches to dealing with typographical errors in character strings. A phonetic function maps words in a natural language to strings representing their pronunciation (the phonetic code). The aim is that words which sound similar enough get the same phonetic code. Obviously one needs different phonetic functions for different languages.[4] Package **RecordLinkage** includes the popular Soundex algorithm for English and a German language algorithm introduced by Michael (1999), implemented through functions `soundex` and `pho_h` respectively. The argument `phonetic` of the compare functions controls the application of the phonetic function, which defaults to `pho_h` and can be set by argument `phonfun`. Typically, an integer vector is supplied which specifies the indices of the data columns for which a phonetic code is to be computed before comparison with other records. Note that the phonetic function, if requested, is applied before the blocking process, therefore the equality restrictions imposed by blocking apply to phonetic codes. Consider, for example, a call with arguments `phonetic = 1:4` and `blockfld = 1`. In this case, the

[3] See documentation for `compare.*` for details.

[4] For this reason, problems may arise when records in one file stem from individuals from different nationalities.

actual blocking criterion is agreement on phonetic code of the first attribute.

String comparators measure the similarity between strings, usually with a similarity measure in the range $[0,1]$, where 0 denotes maximal dissimilarity and 1 equality. This allows 'fuzzy' comparison patterns as displayed in the following example.[5]

```
> rpairsfuzzy <- compare.dedup(RLdata500,
+   blockfld = c(5, 6), strcmp = TRUE)
> rpairsfuzzy$pairs[1:5, ]

  id1 id2  fname_c1 fname_c2  lname_c1 lname_c2
1 357 414 1.0000000       NA 1.0000000       NA
2 389 449 0.6428571       NA 0.0000000       NA
3 103 211 0.7833333       NA 0.5333333       NA
4   6 328 0.4365079       NA 0.4444444       NA
5  37  72 0.9750000       NA 0.9500000       NA
  by bm        bd is_match
1  1  1 0.7000000       NA
2  1  1 0.6666667       NA
3  1  1 0.0000000       NA
4  1  1 0.0000000       NA
5  1  1 1.0000000       NA
```

Controlling the application of string comparators works in the same manner as for phonetic functions, via the arguments `strcmp` and `strcmpfun`. The algorithms by Winkler (1990) (function `jarowinkler`) and one based on the edit distance by Levenshtein (function `levenshteinSim`) are included in the package. String comparison and phonetic encoding cannot be used simultaneously on one attribute but can be applied to different attributes within one set of comparison patterns, as in the function call `compare.dedup(RLdata500, phonetic = 1:4, strcmp = 5:7)`. We refer to the reference manual for further information.

Stochastic record linkage

Theory

Stochastic record linkage relies on the assumption of conditional probabilities concerning comparison patterns. The probabilities of the random vector $\gamma = (\gamma_1,...,\gamma_n)$ having value $\tilde{\gamma} = (\tilde{\gamma}_1,...,\tilde{\gamma}_n)$ conditional on the match status Z are defined by

$$u_{\tilde{\gamma}} = P(\gamma = \tilde{\gamma} \mid Z = 0),\ m_{\tilde{\gamma}} = P(\gamma = \tilde{\gamma} \mid Z = 1),$$

where $Z = 0$ stands for a non-match and $Z = 1$ for a match. In the Fellegi-Sunter model these probabilities are used to compute weights of the form

$$w_{\tilde{\gamma}} = \log\left(\frac{P(\gamma = \tilde{\gamma} \mid Z = 1)}{P(\gamma = \tilde{\gamma} \mid Z = 0)}\right).$$

These weights are used in order to discern between matches and non-matches.

There are several ways of estimating the probabilities involved in this model. In **RecordLinkage** an EM algorithm is used as a promising method for reliable estimations. The backbone of this algorithm is described by Haber (1984). We extended and implemented this algorithm in C in order to improve its performance. Without a proper stochastic model, the probabilities have to be fixed manually. If only weights are to be computed without relying on the assumptions of probabilities, then simple methods like the one implemented by Contiero et al. (2005) are suitable. Further details of these methods are also found in Sariyar et al. (2009).

Weight calculation based on the EM algorithm and the method by Contiero et al. (2005) are implemented by functions `emWeights` and `epiWeights`. Both take a data set object as argument and return a copy with the calculated weights stored in additional components. Calling `summary` on the result shows the distribution of weights in histogram style. This information can be helpful for determining classification thresholds, e.g. by identifying clusters of record pairs with high or low weights as non-matches or matches respectively.

```
> rpairs <- epiWeights(rpairs)
> summary(rpairs)

Deduplication Data Set

500 records
1221 record pairs

49 matches
1172 non-matches
0 pairs with unknown status


Weight distribution:

[0.15,0.2] (0.2,0.25] (0.25,0.3] (0.3,0.35]
      1011          0         89         30
(0.35,0.4] (0.4,0.45] (0.45,0.5] (0.5,0.55]
        29          8          7          1
(0.55,0.6] (0.6,0.65] (0.65,0.7] (0.7,0.75]
        14         19         10          2
(0.75,0.8]
         1
```

Discernment between matches and non-matches is achieved by means of computing weight thresholds. In the Fellegi-Sunter model, thresholds are computed via specification of destined (and feasible) values of homonym and synonym errors so that the amount of doubtable cases is minimized. In the package three auspicious variants for determining the threshold are implemented. The most common practice is to determine thresholds by clerical review, either a single threshold which separates links and non-links or separate thresholds for links

[5] Blocking is used in this example for the purpose of reducing computation time.

and non-links which define a range of doubtable cases between them. **RecordLinkage** supports this by the function `getPairs`, which shows record pairs aligned in two consecutive lines along with their weight (see the following example). When appropriate thresholds are found, classification is performed with `emClassify` or `epiClassify`, which take as arguments the data set object and one or two classification thresholds.

```
> tail(getPairs(rpairs, 0.6, 0.5))

      Weight  id fname_c1 fname_c2 lname_c1
25 0.5924569 266    KARIN     <NA>     HORN
26           437   KARINW     <NA>     HORN
27 0.5924569 395  GISOELA     <NA>     BECK
28           404   GISELA     <NA>     BECK
29 0.5067013 388   ANDREA     <NA>    WEBER
30           408   ANDREA     <NA>  SCHMIDT
   lname_c2   by bm bd
25     <NA> 2002  6  4
26     <NA> 2002  6  4
27     <NA> 2003  4 16
28     <NA> 2003  4 16
29     <NA> 1945  5 20
30     <NA> 1945  2 20

> result <- epiClassify(rpairs, 0.55)
```

The result is an object of class `"RecLinkResult"`, which differs from the data object in having a component `prediction` that represents the classification result. Calling `summary` on such an object shows error measures and a table comparing true and predicted matching status.[6]

```
> summary(result)

Deduplication Data Set

[...]

46 links detected
0 possible links detected
1175 non-links detected

alpha error: 0.061224
beta error: 0.000000
accuracy: 0.997543


Classification table:

           classification
true status    N    P    L
      FALSE 1172    0    0
      TRUE     3    0   46
```

One alternative to threshold determination by clerical review needs labeled training data on which the threshold for the whole data set is computed by minimizing the number of wrongly classified pairs. After weights have been calculated for these data, the classification threshold can be obtained by calling `optimalThreshold` on the training data object.

The other alternative for determining thresholds is an unsupervised procedure based on concepts of extreme value statistics. A mean excess plot is generated on which the interval representing the relevant area for false match rates is to be determined. Based on the assumption that this interval corresponds to a fat tail of the empirical weights distribution, the generalized Pareto distribution is used to compute the threshold discerning matches and non-matches. Details of this latter procedure will be found in a forthcoming paper which is still under review.

A function `getParetoThreshold` is included in the package which encapsulates the necessary steps. Called on a data set for which weights have been calculated, it brings up a mean excess plot of the weights. By clicking on the graph, the boundaries of the weight range in which matches and non-matches presumably overlap are selected. This area is usually discernible as a relatively long, approximately linear section in the middle region of the mean excess graph. This corresponds to the assumption that the generalized Pareto distribution is applicable. The data sets in the package provide only weak support for this assumption, especially because of their limited size. Figure 1 shows an example plot where the appropriate limits are displayed as dashed lines. The return value is a threshold which can be used with `emClassify` or `epiClassify`, depending on the type of weights. We refer to the package vignette *Classifying record pairs by means of Extreme Value Theory* for an example application.

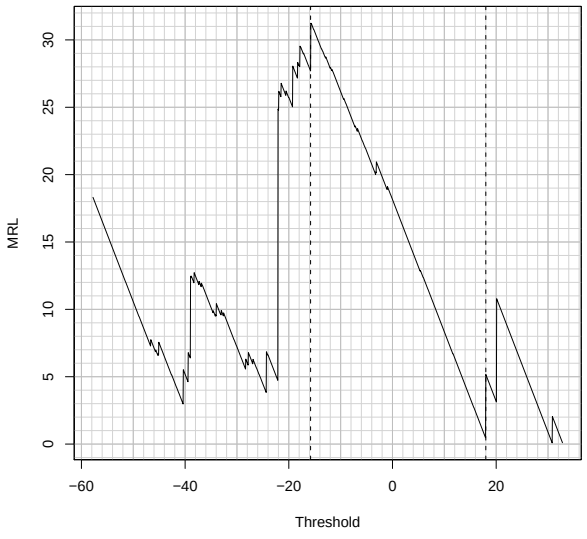


Figure 1: Example mean excess plot of weights with selected limits.

[6] True status is denoted by `TRUE` and `FALSE`, classification result by `"N"` (non-link),`"L"` (link) and `"P"` (possible link). To save space, some output is omitted, marked by '`[...]`' in this and some of the following examples

Machine learning methods

Record Linkage can be understood as a classification problem with comparison pattern γ as input and the matching status variable Z as output. With this view, a vast range of machine learning procedures, such as clustering methods, decision trees or support vector machines, becomes available for deduplicating or linking personal data. The following describes the application of machine learning to record linkage as it is implemented in the package.

Unsupervised classification

Unsupervised classification methods are attractive as they eliminate the necessity to provide representative training data, which can turn out to be a time-consuming task, involving either manual review or the finding of an adequate mechanism to generate artificial data. Motivated by this advantage, unsupervised clustering is incorporated into the package by means of the `classifyUnsup` function, which uses k-means clustering via `kmeans` or bagged clustering via `bclust` from package **e1071** (Dimitriadou et al., 2009). It must be noted that the quality of the resulting classification varies significantly for different data sets and poor results can occur. The following example shows the invocation of k-means clustering and the resulting classification.

```
> summary(classifyUnsup(rpairs, method = "kmeans"))

Deduplication Data Set

[...]

62 links detected
0 possible links detected
1159 non-links detected

alpha error: 0.000000
beta error: 0.011092
accuracy: 0.989353


Classification table:

           classification
true status    N    P    L
      FALSE 1159    0   13
      TRUE     0    0   49
```

Supervised classification

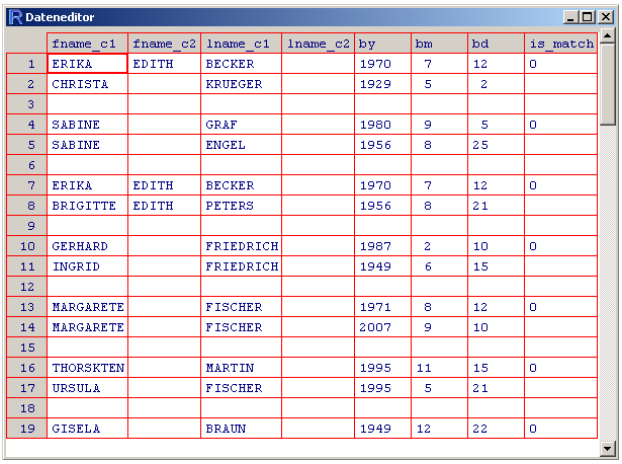
Training data for calibrating supervised classification methods can be obtained in a variety of ways in the context of this package. An important distinction is to be made between cases where additional training data with known true identity status are available and those where a training set has to be determined from the data on which linkage is performed. In the former case, one can provide a distinct set of records which are compared by one of the compare functions (see above) to obtain a training set. In the latter case, two approaches exist, first through what we call a *minimal training set*, second through unsupervised classification.

To construct a minimal training set, comparison patterns in a data set are grouped according to their configuration of agreement values. For every present configuration, one representative is randomly chosen. Naturally, this procedure is only feasible for binary comparisons (agreement or disagreement coded as 1 and 0).

In our experience, the use of supervised classification with a minimal training set can yield results similar to those that might be achieved with randomly sampled training sets of a substantially larger size. Their small magnitude allows minimal training sets to be classified by clerical review with manageable effort.

Two functions in **RecordLinkage** facilitate the use of minimal training sets. A set with the defined properties can be assembled by calling `getMinimalTrain` on a data object. Calling `editMatch` on the result opens an edit window which prints each record pair on two consecutive lines and allows for setting its matching status (as displayed in Figure 2). In the following example, 17 comparison patterns are selected randomly as a minimal training set.

```
> minTrain <- getMinimalTrain(rpairs)
> minTrain <- editMatch(minTrain)
```


Dateneditor

	fname_c1	fname_c2	lname_c1	lname_c2	by	bm	bd	is_match
1	ERIKA	EDITH	BECKER		1970	7	12	0
2	CHRISTA		KRUEGER		1929	5	2	
3								
4	SABINE		GRAF		1980	9	5	0
5	SABINE		ENGEL		1956	8	25	
6								
7	ERIKA	EDITH	BECKER		1970	7	12	0
8	BRIGITTE	EDITH	PETERS		1956	8	21	
9								
10	GERHARD		FRIEDRICH		1987	2	10	0
11	INGRID		FRIEDRICH		1949	6	15	
12								
13	MARGARETE		FISCHER		1971	8	12	0
14	MARGARETE		FISCHER		2007	9	10	
15								
16	THORSKTEN		MARTIN		1995	11	15	0
17	URSULA		FISCHER		1995	5	21	
18								
19	GISELA		BRAUN		1949	12	22	0

Figure 2: Edit window for clerical review

The second approach to obtaining training data when no labelled data are available, provided by function `genSamples`, uses unsupervised clustering in a manner similar to `classifyUnsup`. Instead of directly classifying all the data, a subset is extracted and classified by bagged clustering. It can then be used to calibrate a supervised classifier which is ultimately applied to the remaining set. Arguments to `genSamples` are the original data set, the number of non-matches to appear in the training set and the

desired ratio of matches to non-matches. For example, the call `genSamples(datapairs, num.non = 200, des.mprop = 0.1)` tries to build a training set with 200 non-matches and 20 matches. The return value is a list with two disjoint sets named `train` and `valid`.

In an scenario where different record linkage approaches are evaluated, it is useful to split a data set into training and validation sets. Such a split can be performed by `splitData`. The return value is a list with components `train` and `valid` as in the case of `genSamples`. The most basic usage is to specify a fraction of patterns that is drawn randomly as a training set using the argument `prop`. For example, `splitData(rpairs, prop = 0.1)` selects one tenth of the data for training. By setting `keep.mprop = TRUE` it can be enforced that the original ratio of matches to non-matches is retained in the resulting sets. Another possibility is to set the desired number of non-matches and the match ratio through arguments `num.non` and `mprop` as with `genSamples()`.

Classification functions

All classification methods share two interface functions: `trainSupv` for calibrating a classifier, `classifySupv` for classifying new data. At least two arguments are required for `trainSupv`, the data set on which to train and a string representing the classification method. Currently, the supported methods are:

`"rpart"` Recursive partitioning trees, provided by package **rpart** (Therneau et al., 2009).

`"bagging"` Bagging of decision trees, provided by package **ipred** (Peters and Hothorn, 2009).

`"ada"` Stochastic boosting, provided by package **ada** (Culp et al., 2006).

`"svm"` Support vector machines, provided by package **e1071** (Dimitriadou et al., 2009).

`"nnet"` Single-hidden-layer neural networks, provided by package **e1071** (ibid.).

Of the further arguments, `use.pred` is noteworthy. Setting it to `TRUE` causes `trainSupv` to treat the result of a previous prediction as outcome variable. This has to be used if the training data stem from a run of `genSamples`. Another application is to obtain a training set by classifying a fraction of the data set by the process of weight calculation and manual setting of thresholds. In order to train a supervised classifier with this training set, one would have to set `use.pred = TRUE`.

The return value is an object of class `"RecLinkClassif"`. Classification of new data is carried out by passing it and a `"RecLinkData"` object to `classifySupv`. The following example shows the application of bagging based on the minimal training set `minTrain` from the example above.

```
> model <- trainSupv(minTrain, method = "bagging")
> result <- classifySupv(model, newdata = rpairs)
> summary(result)

Deduplication Data Set

[...]

53 links detected
0 possible links detected
1168 non-links detected

alpha error: 0.020408
beta error: 0.004266
accuracy: 0.995086


Classification table:

           classification
true status    N    P    L
      FALSE 1167    0    5
      TRUE     1    0   48
```

Discussion

During its development, the functionalities of the package **RecordLinkage** have already been used by the authors for a period of about one year, individual functions even before that. Thanks to this process of parallel development and usage, bugs were detected early and new requirements that became apparent were accounted for by implementing new features. Therefore, the package is generally in a usable state. However, there is still potential for future improvements, mainly regarding data handling and performance.

RecordLinkage was developed mainly as a tool for empirical evaluation of record linkage methods, i.e. the main focus of our research was on the performance of particular record linkage methods. Only in one case, the evaluation of a deterministic record linkage procedure used in a German cancer registry, were we actually interested in detecting duplicates. Development of the package followed the needs of this detection process. But many other desirable improvements concerning data processing and input/ouput facilities are needed for a real-world record linkage process, such as:

- The possibility to use a database connection for data input and output.
- Improved print facilities for clerical review, such as highlighting of agreement or disagreement in record pairs.

- The possibility to output a deduplicated data set based on the matching result. This is not a trivial task as it requires choosing the most likely record from each group of matching items which can be accomplished by means of linear programming.

Another important future task is performance enhancement. When handling large data sets of about 10^6 or more record pairs, high memory consumption often leads to errors or destabilizes the computer system R is run on. We are currently working on code changes to avoid unnecessary copying of objects and plan to incorporate these in future releases. Another possibility, which needs further evaluation, is to use more sophisticated ways of storing data, such as a database or one of the various R packages for large data sets. The current disadvantage of R concerning performance and memory can be compensated by an elaborated blocking strategy.

As a general improvement, it is intended to support the usage of algorithms not considered in the package through a generic interface for supervised or unsupervised classification.

We are aware that **RecordLinkage** lacks an important subtask of record linkage, the standardization of records. However, this is an area that itself would need extensive research efforts. Moreover, the appropriate procedures depend heavily on the data to be linked. It is therefore left to the users to tailor standardization methods fitting the requirements of their data. We refer to R's capabilities to handle regular expressions and to the CRAN task view *NaturalLanguageProcessing*[7].

Bibliography

P. Contiero et al. The Epilink record linkage software. *Methods Inf Med.*, 44(1):66–71, 2005.

M. Culp, K. Johnson, and G. Michailidis. *ada: Performs boosting algorithms for a binary response*, 2006. URL http://www.stat.lsa.umich.edu/~culpm/math/ada/img.html. R package version 2.0-1.

E. Dimitriadou, K. Hornik, F. Leisch, D. Meyer, and A. Weingessel. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2009. URL http://CRAN.R-project.org/package=e1071. R package version 1.5-19.

I. Fellegi and A. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

E. Haber. AS207: Fitting a general log-linear model. *Applied Statistics*, 33:358–362, 1984.

J. Michael. Doppelgänger gesucht – ein Programm für kontextsensitive phonetische Textumwandlung. *c't*, 17(25):252–261, 1999. URL http://www.heise.de/ct/ftp/99/25/252/.

A. Peters and T. Hothorn. *ipred: Improved Predictors*, 2009. URL http://CRAN.R-project.org/package=ipred. R package version 0.8-7.

M. Sariyar, A. Borg, and K. Pommerening. Evaluation of record linkage methods for iterative insertions. *Methods Inf Med.*, 48(5):429–437, 2009.

T. M. Therneau, B. Atkinson, and B. Ripley (R port). *rpart: Recursive Partitioning*, 2009. URL http://CRAN.R-project.org/package=rpart. R package version 3.1-45.

W. E. Winkler. String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. In *Proceedings of the Section on Survey Research Methods, American Statistical Association*, pages 354–369, 1990. URL www.amstat.org/sections/srms/proceedings/papers/1990_056.pdf.

Murat Sariyar
Institute of Medical Biostatistics, Epidemiology and Informatics
Mainz
Germany
sariyar@imbei.uni-mainz.de

Andreas Borg
Institute of Medical Biostatistics, Epidemiology and Informatics
Mainz
Germany
borg@imbei.uni-mainz.de

[7] http://cran.r-project.org/view=NaturalLanguageProcessing