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). In versions 0.3.x, this was achieved by blockwise on-demand creation of comparison patterns in an embedded SQLite database (through package RSQLite). Package version 0.4 replaces this mechanism by using file-based data structures from package ff. This approach restricts the amount of data pairs to the available disk space but speeds up execution and facilitates the implementation of methods that need to process the whole set of record pairs (e.g. calculation of optimal classification thresholds).

The interface to the “big data” methods has is compatible to code written for version 0.3.x, so users familiar with these can stick to their existing workflow (unless access to internal structures like object slots is involved). Therefore, the following text sticks to the vignette already included in versions before 0.4 and only technical details are changed to reflect the different implementation.

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 existing 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)
> 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 pairs Wdata
Class: character ffdf ff_vector

Name: WdataInd M U
Class: ff_vector ff_vector ff_vector

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: pairs Wdata WdataInd
Class: ffdf ff_vector ff_vector

Name: M U
Class: ff_vector ff_vector

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
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.

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

```r
# deduplicate dataset with two blocking iterations and string comparison
data(RLdata500)
data(RLdata10000)
 rpairs1 <- RLBigDataDedup(RLdata500, identity = identity.RLdata500, blockfld = list(1,3), strcmp = 1:4)
 # link two datasets with phonetic code, exclude lname_c2
 s1 <- 471:500
 s2 <- sample(1:10000, 300)
 identity2 <- c(identity.RLdata500[s1], rep(NaN, length(s2)))
dataet <- rbind(RLdata500[s1,], RLdata10000[s2,])
 rpairs2 <- RLBigDataLinkage(RLdata500, dataset, identity1 = identity.RLdata500, + identity2 = identity2, phonetic = 1:4, exclude = "lname_c2")
```

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 classifier can only be trained with an object of class "RecLinkData".

```r
# train classifier
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 classification result along with the data object.

```r
showClass("RLResult")
```

Class "RLResult" [package "RecordLinkage"]
Slots:

Name: data prediction
Class: RLBigData ff_vector

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

```r
> getTable(result)

Classification
true status N P L
0 124696 0 4
1 2 0 48
```

```r
> getErrorMeasures(result)

$alpha
[1] 0.04

$beta
[1] 3.207698e-05

$accuracy
[1] 0.9999519

$precision
[1] 0.9230769

$sensitivity
[1] 0.96

$specificity
[1] 0.9999679

$ppv
[1] 0.9230769

$npv
[1] 0.999984
```

### 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):

```r
> rpairs1 <- epiWeights(rpairs1)
> result <- epiClassify(rpairs1, 0.5)
> getTable(result)
```
classification
true status N P L
0 124699 0 1
1 4 0 46

4 Evaluation and results

In addition to `getTable` and `getErrorMeasures`, `getPairs`, which was re-designed 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")
```

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

```
> getFalsePos(result)
```

---

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

```
> getFalsePos(result)
```

---

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

```
> getFalsePos(result)
```

---

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

```
> getFalsePos(result)
```

---

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

```
> getFalsePos(result)
```
> 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  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
>

6