Module 6: Fellegi-Sunter Method Applied to RLdata500

Rebecca C. Steorts

Agenda

- Let's consider the RLdata500 data set and apply Fellegi Sunter to it
- We will first use the RecordLinkage package
- Then we will look at a wrapper function that makes this a bit easier.

Load packages

```r
library(RecordLinkage)
#library(ffbase)
```

RLdata500

```r
data(RLdata500)
head(RLdata500)
```

```
##   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
## 6  JUERGEN     <NA>   FRANKE     <NA> 1929  7  4
```

```r
head(identity.RLdata500)
```

```
## [1]  34  51 115 189  72 142
```

Comparison Vectors

Why do we build record pairs of comparison vectors?

Answer: Reduce the total number of record comparisons.

Comparison Vectors

How do we build record pairs of comparison vectors?

Answer: Use the `compare.dedup` function.

Comparison Vectors

```r
# create comparison vectors
rpairs <- compare.dedup(RLdata500,
                        identity = identity.RLdata500)
```

Comparison Vectors

```r
# inspect comparison vectors
rpairs$pairs[1:5,]
```

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

Blocking

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

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.

Blocking

A blocking specification can be supplied to the `compare` function via the argument `blockfld`.

We will consider a blocking pattern where two records must agree in either the **first component of the first name** or **full date of birth**.

Blocking and Comparison Vectors

```r
# blocking and comparison vectors
rpairs <- compare.dedup(RLdata500,
                        blockfld = list(1,5:7),
                        identity = identity.RLdata500)
```

Blocking and Comparison Vectors

```
# inspect comparison vectors
rpairs$pairs[c(1:3, 1203:1204),]
```

```
##      id1 id2 fname_c1 fname_c2 lname_c1 lname_c2 by bm bd is_match
## 1      1 174        1       NA        0       NA  0  0  0        0
## 2      1 204        1       NA        0       NA  0  0  0        0
## 3      2   7        1       NA        0       NA  0  0  0        0
## 1203 448 497        1       NA        0       NA  0  0  0        0
## 1204 450 477        1       NA        0       NA  0  0  0        0
```

Observe that these records agree on first name but not date of birth (as designed).

String Comparators

Recall that 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.

Examples: Edit and Jaro Winkler

Blocking and String Comparators

```r
# blocking on birth day and month
# use jarowinkler string distance
rpairsfuzzy <- compare.dedup(RLdata500,
                             blockfld = c(5,6),
                             strcmp = TRUE,
                             strcmpfun = jarowinkler)
```

Blocking on **birth day and month** where the **Jaro Winkler** string comparator is used.

Blocking and String Comparators

```r
# inspect first five record pairs
rpairsfuzzy$pairs[1:5,]
```

```
##   id1 id2  fname_c1 fname_c2  lname_c1 lname_c2 by bm bd is_match
## 1   2  43 1.0000000       NA 0.9666667       NA  1  1  1       NA
## 2   4 392 0.5777778       NA 0.4833333       NA  1  1  1       NA
## 3   6 328 0.4365079       NA 0.4444444       NA  1  1  0       NA
## 4   7 129 0.0000000       NA 0.4416667       NA  1  1  0       NA
## 5  11 130 0.4476190       NA 0.0000000       NA  1  1  0       NA
```

Probabilistic record linkage

Probabilistic record linkage relies on the assumption of conditional probabilities concerning comparison patterns.

Recall that we defined the *u* and *m* probabilities previously as the following:

$$u_{\tilde{\gamma}} = P(\gamma = \tilde{\gamma} \mid \text{the records are a match})$$

$$m_{\tilde{\gamma}} = P(\gamma = \tilde{\gamma} \mid \text{the records are not a match}).$$

Probabilistic record linkage

The probabilities of the random vector $\gamma = (\gamma_1, \ldots \gamma_n)$ having value $\tilde{\gamma} = (\tilde{\gamma}_1, \ldots \tilde{\gamma}_n)$ conditional on the match status Z can more precisely be defined as follows:

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

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

where $Z = 0$ stands for a non-match and $Z = 1$ stands for a match.

Probabilistic record linkage

In the Fellegi-Sunter model these probabilities are used to compute weights of the form

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

These weights are used in order to discern between matches and non-matches, where there are several ways to estimate the probabilities/weights.

EM algorithm

The EM algorithm is used typically to estimate the weights, where the backbone of this algorithm is described by Haber (1984).

Weight calculation based on the EM algorithm and the method by Contiero et al. (2005) are implemented by functions `emWeights` and `epiWeights`.

Calling `summary` on the result shows the distribution of weights in histogram style.

EM algorithm

```r
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] (0.35,0.4] (0.4,0.45] (0
##       1011          0         89         30         29          8
## (0.5,0.55] (0.55,0.6] (0.6,0.65] (0.65,0.7] (0.7,0.75] (0.75,0.8]
##          1         14         19         10          2          1
```

Computing Weight Thresholds

Discernment between matches and non-matches is achieved by means of computing weight thresholds.

The function `epiClassify` allows the user to specify a threshold.

Computing Weight Thresholds

```r
result <- epiClassify(rpairs, 0.55)
#summary(result)
```

Objects of `result`

We can look at many objects of `result` which include

- data
- pairs
- frequencies
- type
- Wdatas
- prediction
- threshold

Evaluation Metrics

```
fileDir <- "datasets/"
file.name <- "RLdata10000.csv"
filesDf <- read.csv(paste0(fileDir,file.name), na.strings=c("NA"),
                     stringsAsFactors = FALSE, colClasses = "character"

# ------------------------------ Configuration -----------------------
linkingFields    <- c("fname_c1", "lname_c1", "by", "bm", "bd")
strLinkingFields <- c("fname_c1", "lname_c1")
blockPasses      <- list(c("by"), c("bm", "bd"))
recIdField       <- "rec_id"
entIdField       <- "ent_id"
strDist          <- "levenshtein"
strCutoff        <- 0.70
dataName         <- "RLdata10000"
threshold        <- -10.0
sampleSize       <- c(5, 50) # This give 100 data points
```

Evaluation Metrics

```r
library(ff)
library(ffbase)
source("fs-code/runFS.R")
runFS(filesDf, linkingFields, strLinkingFields, recIdField, entIdField,
      strDist, strCutoff, dataName, threshold)
```

```
## ====================================================================
## ====================================================================
## ====================================================================
## ====================================================================
```

Evaluation Metrics

```r
source("fs-code/evaluationMetrics.R")
source("fs-code/evaluate.R")
evaluate(filesDf, linkingFields, strLinkingFields, recIdField,
         entIdField, blockPasses, strDist, strCutoff, dataName,
         threshold, sampleSize)
```

```
## [1] "SUCCESS"
## [1] "SUCCESS"
## [1] "SUCCESS"
## [1] "SUCCESS"
## [1] "SUCCESS"
```