Record Linkage Exercise

LOAD PACKAGE

To begin, we need to make sure we have the `RecordLinkage` package installed and bring it in.

```
if (!requireNamespace("RecordLinkage", quietly = TRUE)) {
  install.packages('RecordLinkage')
}
library('RecordLinkage', quietly=TRUE)
```

HOW DOES STRCMP WORK?

Before we get into working with an actual dataset, let's take a moment to look at how strcmp works in the RecordLinkage package. You can use both Jaro-Winkler and the Levenshtein distance for string comparison in this package. Additionally, your strings can be adjusted to phonetic forms of the items, to aid in comparison of misspelled words.

Jaro-Winkler comparison takes three weight arguments and r, the Maximum transposition radius. The default values are explicitly employed in the examples below. For more details, this paper is helpful:

Winkler, W.E.: 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 (1990), S. 354-369.

levenshteinDist returns the distance value (an integer), levenshteinSim returns the similarity value (a value between 0 and 1).

The string comparisons are case-sensitive - "R" and "r" have a similarity of 0.

```
str1 = "Katharine"
str2 = "Katherine"
jarowinkler(str1, str2, W_1=1/3, W_2=1/3, W_3=1/3, r=0.5)
```

```
## [1] 0.9555556
```

```
levenshteinSim(str1, str2)
```

```
## [1] 0.8888889
```

```
levenshteinDist(str1, str2)
```

```
## [1] 1
```

```
soundex(str1)
```

```
## [1] "K365"
```

```
soundex(str2)
```

```
## [1] "K365"
```

Comparing a string to a vector of strings:

```
str3 = "Catherine"
jarowinkler(str1, c(str2, str3), W_1=1/3, W_2=1/3, W_3=1/3, r=0.5)
```

```
## [1] 0.9555556 0.8518519
```

```
levenshteinSim(str1, c(str2, str3))
```

```
## [1] 0.8888889 0.7777778
```

```
levenshteinDist(str1, c(str2, str3))
```

```
## [1] 1 2
```

```
soundex(str3)
```

```
## [1] "C365"
```

Case sensitivity:

```
str4 = "KATHARINE"
str5 = "katharine"
jarowinkler(str4, str5, W_1=1/3, W_2=1/3, W_3=1/3, r=0.5)
```

```
## [1] 0
```

```
levenshteinSim(str4, str5)
```

```
## [1] 0
```

```
levenshteinDist(str4, str5)
```

```
## [1] 9
```

```
soundex1 <- soundex(str4)
soundex2 <- soundex(str5)
soundex1
```

```
## [1] "K365"
```

```
soundex2
```

```
## [1] "K365"
```

```
levenshteinSim(soundex1,soundex2)
```

```
## [1] 1
```

LOAD AND SETUP DATA

Now we will read in the data. Note the two slashes are required in Windows file paths. You will need to replace the filepath with where you stored the file on your computer. Also, some code that adjusts the column names for compliance with the big data command requirements is included.

```
sdfilepath = "surveydata.csv"
adfilepath = "admindata.csv"

surveydata <- read.csv(sdfilepath,header=TRUE,sep=",",stringsAsFactors = FALSE)
surveydata <- data.frame(surveydata)

admindata <- read.csv(adfilepath,header=TRUE,sep=",", stringsAsFactors = FALSE)
admindata <- data.frame(admindata)

## Remove periods from column names ##
colnames(surveydata) <- c("matchid","dup","firstname","lastname","maritalstatus","race","dob","ssn","in
colnames(admindata) <- c("matchid","dup","firstname","lastname","maritalstatus","race","dob","ssn","inc
```

```
## DROP EMPTY ROWS ##
surveydata <- surveydata[complete.cases(surveydata[ ,1:2]), ]
admindata <- admindata[complete.cases(admindata[ ,1:2]), ]
```

What are the columns included in the two dataframes?

```
cat("SURVEY DATA VARIABLES\n")
```

```
## SURVEY DATA VARIABLES
```

```
colnames(surveydata)
```

```
##  [1] "matchid"         "dup"             "firstname"       "lastname"
##  [5] "maritalstatus"   "race"            "dob"             "ssn"
##  [9] "income"          "credit_card_num" "city"            "zip"
## [13] "telephone"
```

```
cat("\n\nADMIN DATA VARIABLES\n")
```

```
##
##
## ADMIN DATA VARIABLES
```

```
colnames(admindata)
```

```
##  [1] "matchid"         "dup"             "firstname"       "lastname"
##  [5] "maritalstatus"   "race"            "dob"             "ssn"
##  [9] "income"          "credit_card_num" "city"            "zip"
## [13] "telephone"
```

```
cat("\n\nDATA TYPES\n")
```

```
##
##
## DATA TYPES
```

```
str(surveydata)
```

```
## 'data.frame':    3000 obs. of  13 variables:
##  $ matchid        : int  1003 1011 1013 1017 1028 1029 1031 1036 1040 1041 ...
##  $ dup            : int  1 1 0 0 0 1 0 1 0 1 ...
##  $ firstname      : chr  "Tyron" "Ethan" "Taliah" "Kiara" ...
##  $ lastname       : chr  "Rau" "Claahke" "Staude" "Kellow" ...
##  $ maritalstatus  : chr  "" "Married" "Married" "Married" ...
##  $ race           : chr  "White" "White" "White" "White" ...
##  $ dob            : chr  "5/25/1991" "9/3/1981" "6/24/1954" "2/21/1957" ...
##  $ ssn            : chr  "" "638-11-0983" "176-03-9379" "305-22-0947" ...
##  $ income         : num  NA NA 166135 49913 33235 ...
##  $ credit_card_num: chr  "1915 3160 2544 39912" "2741 2201 8604 48890" "4029 4369 6201 0727" "1116 64
##  $ city           : chr  "Washington DC" "Washington DC" "Washington DC" "Washington DC" ...
##  $ zip            : int  20004 20002 20018 20005 20017 20005 20013 20007 20006 20003 ...
##  $ telephone      : chr  "956-224-0530" "" "313-519-3459" "360-672-5947" ...
```

Create new columns derived from the existing data.

```
# new column with first initial last names concatenated
surveydata$fullname <- paste(substring(surveydata$firstname,1,1), surveydata$lastname, sep = '')
admindata$fullname <- paste(substring(admindata$firstname,1,1), admindata$lastname, sep = '')

# separate day/month/year columns for DOB
```

```
surveydata$dob<- as.Date(surveydata$dob, "%m/%d/%y")
admindata$dob<- as.Date(admindata$dob, "%m/%d/%y")
surveydata$month<- as.numeric(format(surveydata$dob, format = "%m"))
surveydata$day<- as.numeric(format(surveydata$dob, format = "%d"))
surveydata$year<- as.numeric(format(surveydata$dob, format = "%Y"))
admindata$month<- as.numeric(format(admindata$dob, format = "%m"))
admindata$day<- as.numeric(format(admindata$dob, format = "%d"))
admindata$year<- as.numeric(format(admindata$dob, format = "%Y"))

# first three letters of last name (for block field)
surveydata$lastthree <- substring(surveydata$lastname,1,3)
admindata$lastthree <- substring(admindata$lastname,1,3)
str(surveydata)
```

```
## 'data.frame':    3000 obs. of  18 variables:
##  $ matchid        : int  1003 1011 1013 1017 1028 1029 1031 1036 1040 1041 ...
##  $ dup            : int  1 1 0 0 0 1 0 1 0 1 ...
##  $ firstname      : chr  "Tyron" "Ethan" "Taliah" "Kiara" ...
##  $ lastname       : chr  "Rau" "Claahke" "Staude" "Kellow" ...
##  $ maritalstatus  : chr  "" "Married" "Married" "Married" ...
##  $ race           : chr  "White" "White" "White" "White" ...
##  $ dob            : Date, format: "2019-05-25" "2019-09-03" ...
##  $ ssn            : chr  "" "638-11-0983" "176-03-9379" "305-22-0947" ...
##  $ income         : num  NA NA 166135 49913 33235 ...
##  $ credit_card_num: chr  "1915 3160 2544 39912" "2741 2201 8604 48890" "4029 4369 6201 0727" "1116 64
##  $ city           : chr  "Washington DC" "Washington DC" "Washington DC" "Washington DC" ...
##  $ zip            : int  20004 20002 20018 20005 20017 20005 20013 20007 20006 20003 ...
##  $ telephone      : chr  "956-224-0530" "" "313-519-3459" "360-672-5947" ...
##  $ fullname       : chr  "TRau" "EClaahke" "TStaude" "KKellow" ...
##  $ month          : num  5 9 6 2 1 11 11 5 5 8 ...
##  $ day            : num  25 3 24 21 6 27 6 23 27 24 ...
##  $ year           : num  2019 2019 2019 2019 2019 ...
##  $ lastthree      : chr  "Rau" "Cla" "Sta" "Kel" ...
```

Create identity vectors with the matchid variable - a variable that indicates the correct matches of the survey and admin data.

```
identity_survey<- surveydata[ , "matchid"]
identity_admin<- admindata[ , "matchid"]
```

FIRST LINKAGE (just using names)

First, print the column names in the surveydata so we can get the numbers associated for the exclude argument below. (admindata has the same columns)

```
colnames(surveydata)
```

```
##  [1] "matchid"         "dup"             "firstname"       "lastname"
##  [5] "maritalstatus"   "race"            "dob"             "ssn"
##  [9] "income"          "credit_card_num" "city"            "zip"
## [13] "telephone"       "fullname"        "month"           "day"
## [17] "year"            "lastthree"
```

Now we will try our first linkage - we will initially match using only the name fields, using string comparison. We will use city as a blockfield - requiring an exact match on this field - to limit the number of potential matches somewhat. There are only 5 cities represented in the data so this is a conservative blocking. The

exclude arguement includes the column numbers of the fields we wish to not include in the matching process. Therefore it should list all the columns you don't want to include.

We want to use firstname, lastname, and fullname which are columns 3, 4, and 14. We will exclude all other columns (1:2, 5:13, 15:18) For this first run we'll use the default string comparison, Jaro-Winkler.

```
first <- RLBigDataLinkage(surveydata, admindata, identity1=identity_survey, identity2 = identity_admin,
```

The resulting data object contains a row for each potential match. Let's take a look at first 5 pair rows generated. id1 is the ROW number of the record in the first dataset, surveydata; id2 is the row number in the record second dataset, admindata. These are different from the matchid variable included in the dataset. The values under firstname, lastname, and fullname are the Jaro-Winkler distances between the two firstnames, two lastnames, etc. in each pair. is_match is indicated by the identity vectors (the true values), not by running this step of the data linkage procedure.

```
first@pairs[1:5,]
```

```
##   id1 id2 firstname  lastname  fullname     month       day year lastthree
## 1   1   1 0.0000000 0.5111111 0.4722222 0.7777778 0.6666667    1 0.5555556
## 2   1   2 0.6000000 0.4814815 0.4500000 0.7777778 0.7222222    1 0.0000000
## 3   1   3 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000    1 1.0000000
## 4   1   4 0.4555556 0.0000000 0.0000000 0.7777778 0.6666667    1 0.0000000
## 5   1   5 0.4555556 0.5000000 0.4642857 1.0000000 0.8333333    1 0.5555556
##   is_match
## 1        0
## 2        0
## 3        1
## 4        0
## 5        0
```

Let's pick a non-match (rows 1 in both datasets, indicated by the first row of the pairs) and a match (row 1 in the surveydata to row 3 in the admindata, as indicated by the third row in the above pairs) and look at the actual values of those items in the original data to see how the actual data compares to the obtained similarities.

```
cat("FIRST\n")
```

```
## FIRST
```

```
surveydata[1,]$firstname
```

```
## [1] "Tyron"
```

```
admindata[1,]$firstname
```

```
## [1] "Luke"
```

```
admindata[3,]$firstname
```

```
## [1] "Tyron"
```

```
cat("\n\nLAST\n")
```

```
##
##
## LAST
```

```
surveydata[1,]$lastname
```

```
## [1] "Rau"
```

```
admindata[1,]$lastname
```

```
## [1] "Jaric"
```

```
admindata[3,]$lastname
```

```
## [1] "Rau"
```

```
cat("\n\nFULL\n")
```

```
##
##
## FULL
```

```
surveydata[1,]$fullname
```

```
## [1] "TRau"
```

```
admindata[1,]$fullname
```

```
## [1] "LJaric"
```

```
admindata[3,]$fullname
```

```
## [1] "TRau"
```

ASIDE: Alternate ways to perform the linkage

There are alternative ways to obtain the information about whether two records are potential matches. Here we can try the same linkage without strcmp or with using the Levenshtein distance or phonetic match.

```
first_nostring <- RLBigDataLinkage(surveydata, admindata, identity1=identity_survey, identity2 = identi
```

```
first_leven <- RLBigDataLinkage(surveydata, admindata, identity1=identity_survey, identity2 = identity_
```

```
first_phon <- RLBigDataLinkage(surveydata, admindata, identity1=identity_survey, identity2 = identity_a
```

```
first_nostring@pairs[1:5,]
```

```
##   id1 id2 firstname lastname fullname is_match
## 1   1   1         0        0        0        0
## 2   1   2         0        0        0        0
## 3   1   3         1        1        1        1
## 4   1   4         0        0        0        0
## 5   1   5         0        0        0        0
```

```
first_leven@pairs[1:5,]
```

```
##   id1 id2 firstname  lastname  fullname is_match
## 1   1   1 0.0000000 0.2000000 0.1666667        0
## 2   1   2 0.0000000 0.1111111 0.1000000        0
## 3   1   3 1.0000000 1.0000000 1.0000000        1
## 4   1   4 0.0000000 0.1250000 0.1111111        0
## 5   1   5 0.1666667 0.1666667 0.1428571        0
```

```
first_phon@pairs[1:5,]
```

```
##   id1 id2 firstname lastname fullname is_match
## 1   1   1         0        0        0        0
## 2   1   2         0        0        0        0
## 3   1   3         1        1        1        1
```

```
## 4   1   4         0        0        0        0
## 5   1   5         0        0        0        0
```

BACK to the first linkage process: EM Weights

Remember, previously we ran the following code to generate the Jaro-Winkler similarity values.

```
first <- RLBigDataLinkage(surveydata, admindata, identity1=identity_survey, identity2 = identity_admin,
```

The next step in the process of performing the linkage the obtain the summary weights for these potential links. In order to do this, we need to tell the EM algorithm which cutoff we want to use for the similarity distances to convert those numbers into either 0 or 1.

"The appropriate value of cutoff depends on the choice of string comparator. The default is adjusted to jarowinkler, a lower value (e.g. 0.7) is recommended for levenshteinSim" (Borg and Sariyar, 2015).

The cutoff is used to convert each of the distance values to 0/1 binary values.

```
firstw <- emWeights(first, cutoff = 0.95) #you set cutoff - it can be higher or lower if you wish
```

```
## Count pattern frequencies...
```

```
## Run EM algorithm...
```

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

We'll print the summary of the linkage process, the first 5 weights (corresponding to the rows we looked at above), and a summary of the distribution of the weights.

```
print(firstw)
```

```
##
## Linkage Data Set for large number of data
##
## 3000 records in first data set
## 13551 records in second data set
## 8130656 record pairs
```

```
firstw@Wdata[1:5,]
```

```
## [1] -13.48970 -13.48970  46.86052 -13.48970 -12.51726
```

```
summary(firstw@Wdata[])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -13.77  -13.49  -13.49  -13.14  -13.49   46.86
```

Using those weights we will classify the links into matches and nonmatches, which is required for the prediction process.

You use thresholds based on the weight distribution (above) for the classifications. A pair with weight w is classified as a link if w >= threshold.upper, as a possible link if threshold.upper <= w >= threshold.lower and as a non-link if w < threshold.lower.

To get an idea of what the weights look like, we'll first get our pairs and inspect their weights. We know there are a bunch of weights below 0, but they're likely non-matches. To limit processing time and size of objects, we'll only display with a weight higher than 0 here as they are most likely to be potential links.

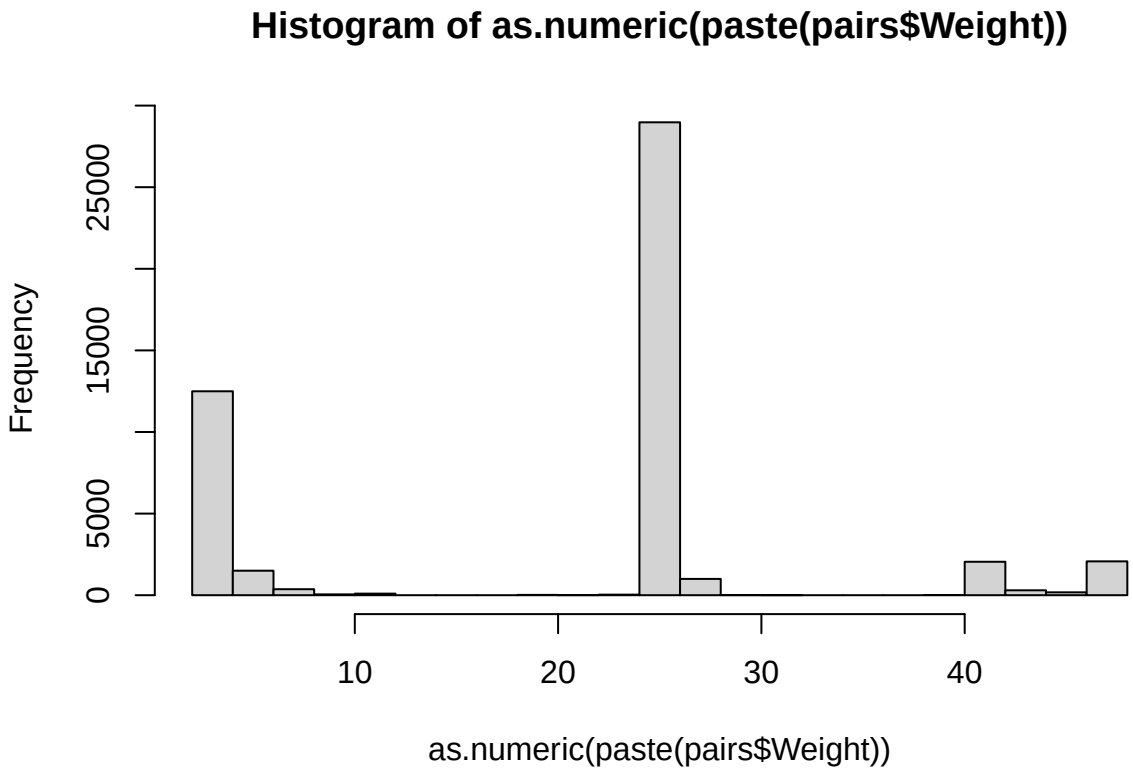
```
pairs=getPairs(firstw, min.weight=0, single.rows=TRUE)
```

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

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x, na.rm = na.rm): no non-missing arguments to max; returning -
## Inf
```

```
hist(as.numeric(paste(pairs$Weight)))
```



For our first classification, we'll specify only one threshold, the threshold.upper, the level in which at or above that weight pairs are considered matches. In this case there will be no possible links, only Links (L) and Non-Links (NL).

We'll start with setting the threshold at the mean of the weights (-19.68), indicated in the distribution summary above. This is an impractical value, however, because anything below 0 is very likely not a match.

```
first_class1 <- emClassify(firstw, threshold.upper = -19.68)
```

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x, na.rm = na.rm): no non-missing arguments to max; returning -
## Inf
```

```
getTable(first_class1)
```

```
## < table of extent 0 x 0 >
```

Above you can see the classification table that shows the true status of the matches (from our identity vectors) and the classification (N=non-link, P=possible link, L=Link)

We can also get the error measures for this linkage. Below you will see that the sensitivity is high (you're getting almost all of the true matches as links), but your precision is low (you're getting over 66k of non-matches as links).

```
getErrorMeasures(first_class1)
```

```
## $alpha
## [1] 0
```

```
##
## $beta
## [1] 1
##
## $accuracy
## [1] 0.0003341674
##
## $precision
## [1] 0.0003341674
##
## $sensitivity
## [1] 1
##
## $specificity
## [1] 0
##
## $ppv
## [1] 0.0003341674
##
## $npv
## [1] NaN
```

Let's set our threshold slightly higher and see how it affects our classification. Let's try 0.

```
first_class2 <- emClassify(firstw, threshold.upper = 0)
```

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x, na.rm = na.rm): no non-missing arguments to max; returning -
## Inf
```

```
getTable(first_class2)
```

```
## < table of extent 0 x 0 >
```

```
getErrorMeasures(first_class2)
```

```
## $alpha
## [1] 0.0493191
##
## $beta
## [1] 0.005731834
##
## $accuracy
## [1] 0.9942536
##
## $precision
## [1] 0.05253096
##
## $sensitivity
## [1] 0.9506809
##
## $specificity
## [1] 0.9942682
##
## $ppv
## [1] 0.05253096
```

```
##
## $npv
## [1] 0.9999834
```

This greatly improves our precision with a slight loss to sensitivity, but in practically, half of the things we have classified as links are in actuality non-matches. We should try a higher threshold to further increase our precision.

Going back to our histogram, we see that large number of the links have weights over 25, with about 3000ish with weights over 30. Lets try setting two thresholds, with our lower threshold at 25 and our upper threshold at 30 and see how it effects our outcome stats.

```
first_class3 <- emClassify(firstw, threshold.lower = 25, threshold.upper = 30)
```

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x, na.rm = na.rm): no non-missing arguments to max; returning -
## Inf
```

```
getTable(first_class3)
```

```
## < table of extent 0 x 0 >
```

```
getErrorMeasures(first_class3)
```

```
## $alpha
## [1] 0.1080166
##
## $beta
## [1] 0.0002761989
##
## $accuracy
## [1] 0.9996886
##
## $precision
## [1] 0.5136541
##
## $sensitivity
## [1] 0.8919834
##
## $specificity
## [1] 0.9997238
##
## $ppv
## [1] 0.5136541
##
## $npv
## [1] 0.9999647
```

This appears to greatly improve the precision of our linkages (many fewer false positives) with only a slight decrease in sensitivity. But in this case we have about 2300 possible links which would have to be manually inspected, a time consuming and labor intensive process.

Optimal Threshold

The package does have a method one can use to obtain an "optimal" threshold. The authors of the package describe how the optimal value is obtained as thus: "For the following, it is assumed that all records with weights greater than or equal to the threshold are classified as links, the remaining as non-links. If no

further arguments are given, a threshold which minimizes the absolute number of misclassified record pairs is returned"

Basically, the optimal threshold is guided by maximizing accuracy, which may not be the best "optimal" threshold for practical work with large numbers of possible matches.

```
threshold <-optimalThreshold(firstw)
```

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x, na.rm = na.rm): no non-missing arguments to max; returning -
## Inf
```

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

```
threshold
```

```
## [1] 41.39764
```

The optimal threshold we obtain is -2.2. As we've seen above, anything below zero is likely impractical. Additionally, while accuracy is maximized, we have a large number of possible links (over 8 million) and the accuracy measure is driven up by the large number of true negatives. (see classification table below)

```
first_class4 <- emClassify(firstw, threshold.upper = threshold)
```

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x, na.rm = na.rm): no non-missing arguments to max; returning -
## Inf
```

```
getTable(first_class4)
```

```
## < table of extent 0 x 0 >
```

```
getErrorMeasures(first_class4)
```

```
## $alpha
## [1] 0.1288185
##
## $beta
## [1] 3.100417e-05
##
## $accuracy
## [1] 0.999926
##
## $precision
## [1] 0.9037801
##
## $sensitivity
## [1] 0.8711815
##
## $specificity
## [1] 0.999969
##
## $ppv
## [1] 0.9037801
##
## $npv
## [1] 0.9999569
```

SECOND LINKAGE (using more variables)

We can also improve the quality of the record linkage by using more variables in our linkage process. Let's refresh ourselves on the columns available.

```
colnames(surveydata)
```

```
##  [1] "matchid"         "dup"             "firstname"       "lastname"
##  [5] "maritalstatus"   "race"            "dob"             "ssn"
##  [9] "income"          "credit_card_num" "city"            "zip"
## [13] "telephone"       "fullname"        "month"           "day"
## [17] "year"            "lastthree"
```

This time we want to use the names, dob (with month day year variables), zip, and ssn. So we will include columns 3,4,8,12,14,15,16,17. We will keep city as the blockfield. Note that we specify only the variables of type "chr" in the strcmp argument, of which ssn is one.

```
second <- RLBigDataLinkage(surveydata, admindata, identity1=identity_survey, identity2 = identity_admin
```

```
second@pairs[1:5,]
```

```
##   id1 id2 firstname  lastname ssn zip  fullname month day year is_match
## 1   1   1 0.0000000 0.5111111   0   0 0.4722222     0   0    1        0
## 2   1   2 0.6000000 0.4814815   0   0 0.4500000     0   0    1        0
## 3   1   3 1.0000000 1.0000000   0   1 1.0000000     1   1    1        1
## 4   1   4 0.4555556 0.0000000   0   0 0.0000000     0   0    1        0
## 5   1   5 0.4555556 0.5000000   0   0 0.4642857     1   0    1        0
```

And then get the weights

```
secondw <- emWeights(second, cutoff = 0.95) #you set cutoff - it can be higher or lower if you wish - b
```

```
## Count pattern frequencies...
```

```
## Run EM algorithm...
```

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

```
print(secondw)
```

```
##
## Linkage Data Set for large number of data
##
## 3000 records in first data set
## 13551 records in second data set
## 8130656 record pairs
```

```
summary(secondw@Wdata[])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -14.74  -13.79  -13.79  -12.96  -13.79   65.73
```

Now use the classifier to classify links vs. non links.

Let's again look at the weights to pick our potential weight threshold.

```
pairs=getPairs(secondw, min.weight=0, single.rows=TRUE)
```

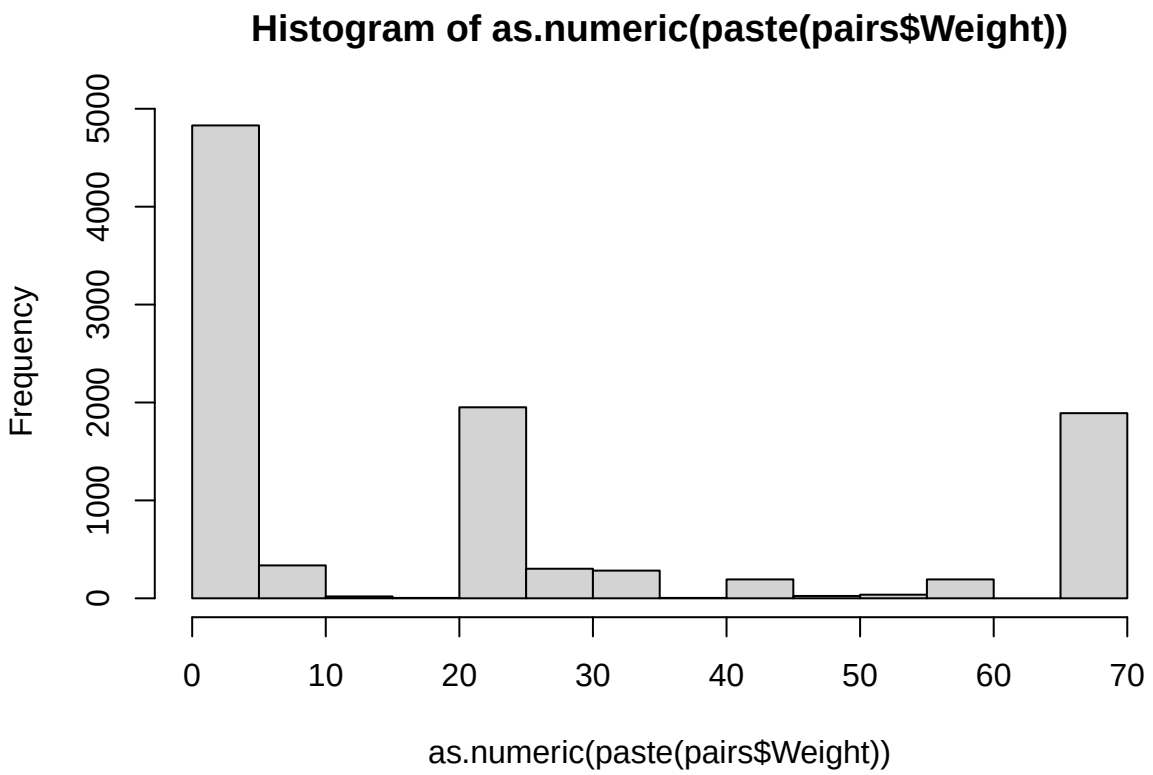
```
## ================================================================================
```

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x, na.rm = na.rm): no non-missing arguments to max; returning -
```

```
## Inf
```

```
hist(as.numeric(paste(pairs$Weight)))
```



The mean of the weights is now about -13, but we'll first try a weight of 0 since anything below 0 is not likely to be an actual link.

```
second_class1 <- emClassify(secondw, threshold.upper = 0)
```

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x, na.rm = na.rm): no non-missing arguments to max; returning -
## Inf
```

```
getTable(second_class1)
```

```
## < table of extent 0 x 0 >
```

```
getErrorMeasures(second_class1)
```

```
## $alpha
## [1] 0.004416636
##
## $beta
## [1] 0.0009057647
##
## $accuracy
## [1] 0.9990931
##
## $precision
## [1] 0.2686997
##
## $sensitivity
```

```
## [1] 0.9955834
##
## $specificity
## [1] 0.9990942
##
## $ppv
## [1] 0.2686997
##
## $npv
## [1] 0.9999985
```

With this threshold we get very few false negatives, but now a large number of false positives. So our sensitivity is very high, but our precision is very low. These are the tradeoffs we deal with in setting the appropriate thresholds, especially when you don't know the true status of the matches.

Judging by the histogram, let's try one final set of weight thresholds. Let's set 20 as the lower threshold and 40 as the upper threshold.

```
second_class2 <- emClassify(secondw, threshold.lower = 20, threshold.upper = 40)
```

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x, na.rm = na.rm): no non-missing arguments to max; returning -
## Inf
```

```
getTable(second_class2)
```

```
## < table of extent 0 x 0 >
```

```
getErrorMeasures(second_class2)
```

```
## $alpha
## [1] 0.04141041
##
## $beta
## [1] 0
##
## $accuracy
## [1] 0.9999876
##
## $precision
## [1] 1
##
## $sensitivity
## [1] 0.9585896
##
## $specificity
## [1] 1
##
## $ppv
## [1] 1
##
## $npv
## [1] 0.9999876
```

In this case we get no false positives, so our precision is 1. But we have about 14% of our true links classified as probable or non links,

GET PAIRS AND EXPORT DATASET

If you were going to use your final set of matched pairs for further analysis you would want to use getPairs to obtain just the records for the records classified as links. You can also export them out of R into a csv file.

```
finalpairs = getPairs(second_class2, filter.link="link", single.rows=TRUE)
```

```
## Warning in min(x, na.rm = na.rm): no non-missing arguments to min; returning Inf
```

```
## Warning in max(x, na.rm = na.rm): no non-missing arguments to max; returning -
## Inf
```

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

```
csvfilepath="finalpairs.csv"
write.csv(finalpairs, file=csvfilepath)
```

REFERENCE (Package Documentation)

Andreas Borg and Murat Sariyar (2015). RecordLinkage: Record Linkage in R.