svisits: finding HIV transmission and serosorting pairs using shared clinic visit dates

Alex Marzel, Teja Turk

This package helps to find some stable HIV-infected partnerships in cohort studies based on the assumption that some patients frequently attend the clinical visits together. These pairs are useful for biological and epidemiological research.

Package installation:

```r
install.packages("https://github.com/alexmarzel/svisits/raw/master/svisits_1.1.tar.gz",
                 repos = NULL, type = "source")
library(svisits)
set.seed(14051948)
```

The following paragraphs explain step-by-step how the **svisits** package can be used to find stable HIV-infected partnerships. All the steps can be also performed with one function call, which is demonstrated below.

First let us load the database with visit dates in a long format (each row represents a visit of a particular person) and transform it to the appropriate format. The following simulated dataset contains 20 transmission pseudo pairs. The data were simulated based on the visits distribution in the SHCS.

```r
data("simulated_data")
# prepare the database
db_dates <- prepare_db(your_database = simulated_data,
                       ids_column = "subject",
                       dates_column = "sim_dates")
head(db_dates)
```

```
##   subject      dates
## 1       1 1993-10-02
## 2       1 1993-12-24
## 3       1 1994-03-16
## 4       1 1994-05-27
## 5       1 1994-08-21
## 6       1 1994-11-06
```

Deconstruct the dates into 3 months periods:

```r
# extract month, day and year of each clinic visit:
db_dates <- cbind(db_dates, month.day.year(db_dates$dates))
# deconstruct the dates into 3 months periods
periodMonths <- 3
db_dates <- cbind(db_dates,
                  fupdatePeriod=db_dates$year+round(floor((db_dates$month-1)/periodMonths)
                                                    *periodMonths/12,
                                                    digits=2))
```

Get unadjusted, observed shared visits. It can take some time.

```r
unadjusted_observed_pairs <- get_observed_pairs(db_dates)
head(unadjusted_observed_pairs)
```

```
## allPairs
## 1000_1163 1000_1168 1000_1258 1000_1310 1000_1347 1000_1388
##          1          1          1          1          1          1
```

Descriptive statistics of shared visits for pairs that shared at least a single visit:

```
summary(as.numeric(unadjusted_observed_pairs))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   1.321   1.000  24.000
```

Prepare the ids:

```
ids_table <- table(db_dates$subject)
ids <- data.table(ids = as.character(names(ids_table)),
                  N_visits = as.character(as.numeric(ids_table)))
setkey(ids, "ids")
```

Chances that two unrelated cohort members share a visit are increasing with the overall number of visits of each member of the pair. To account for this we have to adjust the number of shared visits S within each pair. This will return the adjusted number of shared visits S' for the observed, unshuffled data

$$S' = S - \frac{\log\left[\binom{\min(T_a, T_b)}{S}\right]}{\log\left[75\right]},$$

where T_a and T_b denote the number of visits of each member of the pair. The adjusted number of visits is under the column "Corrected":

```
adjusted_observed <- adjust_visits(unadjusted_pairs = unadjusted_observed_pairs,
                                   ids = ids,
                                   prob = 1/75)
head(adjusted_observed)
```

```
##   allPairs Freq id_1 id_2 N_visits.x N_visits.y Corrected
## 1     1_10    1    1   10         46         68 0.1132248
## 2     4_10    1    4   10         39         68 0.1514599
## 3     8_10    1    8   10         10         68 0.4666841
## 4     9_10    1    9   10         12         68 0.4244555
## 5 161_1000    1  161 1000         17         11 0.4446087
## 6 176_1000    1  176 1000         11         11 0.4446087
```

Threshold for the number of shared visits

Shuffling

Due to the multiple comparisons problem and strongly simplifying assumptions about the uniform distribution of visits, the corrected number of shared visits is not an optimal test statistic. Instead, the visits are first shuffled within each quarter (such that the original distribution of the number of visits per individual is preserved) and the number of randomly collided shared visits per pair is counted and penalized using "adjust_visits". In other words, the observed visit dates from a given quarter are randomly re-assigned between the patients that attended the clinic during this quarter.

Define the number of desired shuffling simulations (ideally > 100, but it will take some time) and run. It will take some time to finish.

```
n_shuffl <- 10
Shuffling_simulation_output <- replicate(n_shuffl,
                                         Shuffling_simulation(db_dates,
                                                              ids = ids,
                                                                   adjusted_observed),
                                                simplify=F)

# inspect the false-positive thresholds
head(Shuffling_simulation_output)
```

```
## [[1]]
## [1] 0.9394822 0.7810526 0.3095238 0.0000000 0.0000000 0.0000000 0.0000000
## [8] 0.0000000
##
## [[2]]
## [1] 0.9434465 0.7515789 0.2857143 0.0000000 0.0000000 0.0000000 0.0000000
## [8] 0.0000000
##
## [[3]]
## [1] 0.9418979 0.6905263 0.1904762 0.0000000 0.0000000 0.0000000 0.0000000
## [8] 0.0000000
##
## [[4]]
## [1] 0.9389866 0.6673684 0.2619048 0.0000000 0.0000000 0.0000000 0.0000000
## [8] 0.0000000
##
## [[5]]
## [1] 0.9485258 0.6547368 0.1428571 0.0000000 0.0000000 0.0000000 0.0000000
## [8] 0.0000000
##
## [[6]]
## [1] 0.92758920 0.65263158 0.30952381 0.04761905 0.00000000 0.00000000
## [7] 0.00000000 0.00000000
```

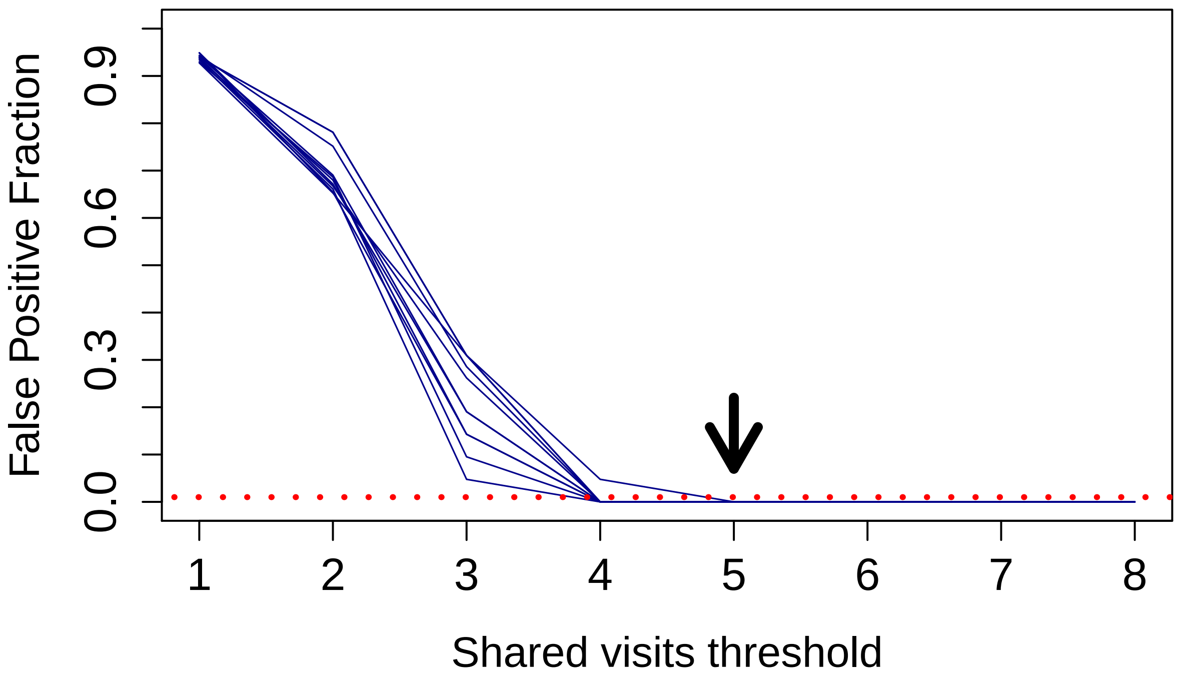
Extract the False-Positive thresholds:

```
# Thresholds
for_thresholds <- do.call("rbind", Shuffling_simulation_output)

# Take max FP value for each position from all the simulations.
# One can also take mean or median insted of maximum
for_thresholds_max <- apply(X = for_thresholds,
                            MARGIN = 2,
                            max,na.rm = TRUE)

# find lowest threshold below alpha=0.01
Thresholds_lowest <- match(for_thresholds_max[for_thresholds_max<0.01][1],
                           for_thresholds_max)
```

Plot the simulations and the False-Positive threshold (indicated by an arrow):

```
max_threshold <- length(Shuffling_simulation_output[[1]])
plot(x=1:max_threshold,
     y=unlist(Shuffling_simulation_output[[1]]),
     xlab="Shared visits threshold", ylab="False Positive Fraction",
     type="l", col="darkblue", yaxp=c(0.0,1,10), lwd=0.8, main="",
     xaxp=c(1,12,11), cex.lab=1.3, cex.axis=1.4, cex.main=1.3, ylim=c(0,1))
arrows(x0 = Thresholds_lowest, x1=Thresholds_lowest,
       y0=0.22, y1=0.07, cex=3, lwd=5)
```

```r
invisible(lapply(X=1:n_shuffl,
                 function(X) {
                   lines(x=1:max_threshold, y=unlist(Shuffling_simulation_output[[X]]),
                         col="darkblue", lwd=0.8 )
                   }))
lines(c(-100,100), 0.01*c(1,1), lty=3, col="red", lwd=3)
```



```r
Thresholds_lowest
```

```
## [1] 5
```

```r
# This is the cutoff for the number of adjusted visits
```

Extract the pairs above the threshold:

```r
select_shuffled <- adjusted_observed[which(adjusted_observed$Corrected>Thresholds_lowest),]

# We found the 20 pairs
length(select_shuffled[,1])
```

```
## [1] 20
```

Alternative (and much faster approach): Bonferroni correction

Predict the probabilities and than adjust for multiple testing using Bonferroni:

```r
# alpha 0.01, one false-positive pair
Bonferroni_m_output <- Bonferroni_m(unadjusted_observed_pairs,
                                    ids = ids,
                                    prob = 1/75,
                                    alpha =0.01)
length(Bonferroni_m_output[,1])
```

```
## [1] 21
```

```
# alpha 0.001, only 20 true pairs
Bonferroni_m_output <- Bonferroni_m(unadjusted_observed_pairs,
                                    ids = ids,
                                    prob = 1/75,
                                    alpha =0.001)
length(Bonferroni_m_output[,1])
```

```
## [1] 20
```

Summary: one-step-approach

The above pairs can be alternatively found with one function call:

```
library(svisits)
data("simulated_data")
set.seed(14051948)

# with shuffling and Bonferroni for alpha=0.01
pairs <- find_pairs(simulated_data,
                    ids_column = "subject",
                    dates_column = "sim_dates",
                    shuffling = TRUE,
                    n_shuffl = 10)
head(pairs$Shuffled_pairs)
```

```
##         allPairs Freq id_1 id_2 N_visits.x N_visits.y Corrected
## 62746   328_1453   22  328 1453         38         44  16.48171
## 65536  1173_1468   24 1173 1468         70         77  14.10959
## 85836   482_1587   21  482 1587         77         66  11.96045
## 101885 1504_1696   17 1504 1696         44         37  11.55935
## 141792   72_1898   16   72 1898         33         39  11.16443
## 142396  793_1900   18  793 1900         44         37  12.53495
```

```
# show similarity
identical(select_shuffled, pairs$Shuffled_pairs)
```

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

```
# with only Bonferroni for alpha=0.001
pairs_Bonferroni <- find_pairs(simulated_data,
                    ids_column = "subject",
                    dates_column = "sim_dates",
                    alpha = 0.001,
                    shuffling = FALSE)
head(pairs_Bonferroni$Bonferroni_pairs)
```

```
##         allPairs Freq id_1 id_2 N_visits.x N_visits.y Prob_for_Bonferr
## 62746   328_1453   22  328 1453         38         44     1.005805e-31
## 65536  1173_1468   24 1173 1468         70         77     1.885820e-27
## 85836   482_1587   21  482 1587         77         66     2.046927e-23
## 101885 1504_1696   17 1504 1696         44         37     1.617750e-22
## 141792   72_1898   16   72 1898         33         39     9.266487e-22
## 142396  793_1900   18  793 1900         44         37     2.429054e-24
##            BP      ltp
## 62746   62746 30.99749
## 65536   65536 26.72450
## 85836   85836 22.68890
## 101885 101885 21.79109
## 141792 141792 21.03308
## 142396 142396 23.61456
```

```
# show similarity
identical(Bonferroni_m_output, pairs_Bonferroni$Bonferroni_pairs)
```

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