DP Second Laboratory

Teresa Ricciardi Albert Bertran

2023-04-10

```
library(sdcMicro)
data("free1") # loads the dataset
```

1.PRAM (Post Randomization Method)

a) Use table() to obtain the frequencies for each category of the MARSTAT variable of the free1 dataset. Define your own P matrix (remember the sum of the rows must be 1) and calculate the theoretical expected frequency for each category after "praming" the MARSTAT variable. Are the expected values of the frequencies after "praming" similar to the original ones? First, we create a newdataset from the "free1" dataset and we check the different values MARSTAT has.

```
newdataset<-free1
v <- table(newdataset[,"MARSTAT"])
v
```

```
##
##    1    2    3    4
## 2547  162  171 1120
```

We now create a P matrix to use for pramming. The values used are just the ones in the slides of the session.

```
mdat <- matrix(c(0.5,0.4,0.1,0,
                 0.1,0.6,0.2,0.1,
                 0.1,0.3,0.6,0,
                 0,0.1,0.2,0.7), nrow = 4, byrow = TRUE,
               dimnames = list(c("single","married","divorced","widow"),
                               c("single","married","divorced","widow")))
mdat
```

```
##          single married divorced widow
## single      0.5     0.4      0.1   0.0
## married     0.1     0.6      0.2   0.1
## divorced    0.1     0.3      0.6   0.0
## widow       0.0     0.1      0.2   0.7
```

Now, using crossprod we can compute the new values, noting that they differ a lot from the original ones, so the matrix used is not ideal.

```
crossprod(v, mdat)
```

```
##      single married divorced widow
## [1,] 1306.8  1279.3    613.7 800.2
```

We now convert the dataset to a dataframe to access it in the sdcApp.

```
newdataset<-as.data.frame(newdataset)
```

b) Check your result using sdcApp(). Setup a DSC problem using the free1 dataset selecting REGION, SEX and AGE as categorical variables and MARSTAT as the variable to be "pramed". To obtain always the same result with probabilistic methods use the same seed. (Don't worry about parameter alpha). Obtain the frequencies for each category of the variable MARSTAT ("Microdata/Explore variables" option). Use PRAM (expert) option in the "Anonymize" menu to "pram" MARSTAT with the matrix that you have defined in the previous section. Check the new frequencies ("Anonymize/Explore variables" option).

```
#sdcApp()
```

Frequencies before applying PRAM using our custom matrix:

Marstat	Frequency	Percentage
1	2547	63.68
2	162	4.05
3	171	4.28
4	1120	28
Sum	4000	100

Frequencies after applying PRAM using our custom matrix:

Marstat	Frequency	Percentage
1	1357	33.92
2	1224	30.60
3	633	15.82
4	786	19.65
Sum	4000	100

As can be seen, the values differ a lot from the original ones, so the matrix is not ideal as we figured out already in the previous exercise.

c) Undo the last step and go to the "Anonymize/PRAM (simple)" option to create an invariant probability transition matrix and "pram" the variable MARSTAT (use the default values pd=0.8 and alpha=0.5). Create a table comparing the frequencies and percentages of the MARSTAT variable before and after "praming" (use "Explore variables" in "Microdata" and "Anonymize" menus). Are the frequencies similar?

Frequencies and percentages before applying pram simple:

Marstat	Frequency	Percentage
1	2547	63.68
2	162	4.05
3	171	4.28
4	1120	28
Sum	4000	100

Frequencies and percentages after applying pram simple:

Marstat	Frequency	Percentage
1	2538	63.45
2	166	4.15
3	176	4.4
4	1120	28
Sum	4000	100

In this case, since the matrix used is way more accurate to the data we are using, we can see that the percentages and frequencies are almost the same before and after applying PRAM, with some small changes.

d) Create an sdcObject like the one in the previous sections and "pram" the MARSTAT variable. Get the PRAM array from the "pram" slot of the sdcObject and perform the following calculation:

- Transpose the matrix and calculate the eigenvalues and the eigenvectors
- Check that 1 is one of the eigenvalues
- Normalize its associated eigenvector so that the sum of its components is 1.
- Compare the eigenvector with the percentages of the original values of MARSTAT. Drawn your own conclusions.

Note that even though we run the commands in the console we attach them here.

Creation of the problem.

```
inputdata <- readMicrodata(path="newdataset", type="rdf", convertCharToFac=FALSE, drop_all_missings=FALS

inputdataB <- inputdata

inputdataB <- varToFactor(obj=inputdataB, var=c("SEX"))
inputdataB <- varToFactor(obj=inputdataB, var=c("AGE"))
inputdataB <- varToFactor(obj=inputdataB, var=c("REGION"))
inputdataB <- varToFactor(obj=inputdataB, var=c("MARSTAT"))

sdcObj <- createSdcObj(dat=inputdataB,
                       keyVars=c("REGION","SEX","AGE"),
                       numVars=NULL,
                       weightVar=NULL,
                       hhId=NULL,
                       strataVar=NULL,
                       pramVars=c("MARSTAT"),
                       excludeVars=NULL,
                       seed=500,
                       randomizeRecords=FALSE,
                       alpha=c(1))
```

Setting the transition matrix.

```
sdcObjp <- pram(sdcObj, variables=c("MARSTAT"), pd=0.8, alpha=0.5)
P <- sdcObjp@pram$params$MARSTAT$Rs
P
```

```
##            1           2           3          4
## 1 0.97657259 0.004492974 0.003976878 0.01495756
## 2 0.07063954 0.826899555 0.007072377 0.09538853
## 3 0.05923456 0.006700147 0.851085505 0.08297979
## 4 0.03401509 0.013797269 0.012669236 0.93951840
```

Transposing the matrix and obtaining the eigenvectors and eigenvalues.

```
transpose <- t(P)
transpose
```

```
##             1           2           3          4
## 1 0.976572589 0.070639540 0.059234556 0.03401509
## 2 0.004492974 0.826899555 0.006700147 0.01379727
## 3 0.003976878 0.007072377 0.851085505 0.01266924
## 4 0.014957559 0.095388528 0.082979792 0.93951840
```

```
eigen(transpose)
```

```
## eigen() decomposition
## $values
## [1] 1.0000000 0.9378961 0.8410547 0.8151252
##
## $vectors
##            [,1]        [,2]       [,3]        [,4]
## [1,] 0.91214222  0.76054906 -0.1473764 -0.23432656
## [2,] 0.05801611 -0.05309503 -0.1608589  0.75123525
## [3,] 0.06123923 -0.06346812  0.8267617  0.09307046
## [4,] 0.40109905 -0.64398591 -0.5185264 -0.60997914
```

Normalize of the first eigenvector, the one associated with the 1 eigenvalue.

```
eig <- eigen(transpose)

eigvec <- eig$vectors[,1]
eigvec <- eigvec / sum(eigvec)
eigvec
```

```
## [1] 0.63675 0.04050 0.04275 0.28000
```

As can be seen, the percentages, are almost the same so the praming procedure did not have a significant impact on the distribution of categories.

Now let's execute again sdcApp() with pd=0.2 and alpha=0.9 and check the results.

The following tables represents before and after applying PRAM(Simple) with the parameters above. Note that the numbers of MARSTAT represents the following status:

- 4 -> Single
- 3 -> Widle
- 2 -> Divorced
- 1 -> Married

The following is the original values of the range (15-18):

Age/Marstat	1	2	3	4
15	1	0	0	55
16	1	0	0	72
17	2	0	0	59
18	0	0	0	61

And after applying PRAM we obtain:

Age/Marstat	1	2	3	4
15	32	2	2	20
16	44	4	2	23
17	32	3	5	21
18	42	0	1	18

We can see that the result is far away from the original values. The most significant case is that, there are 32 15-years old persons married, which is quite atypical among young people. In order to avoid this problem, we could modify the transition matrix so not that much people turn into married when single.

2.Microaggregation

a) Compare the univariate, multivariate simple, and mdav microaggregation algorithms.

```
dataset32<-free1

dataset32<-as.data.frame(dataset32)

sdc <- createSdcObj(
  dat = dataset32,
  keyVars = c("REGION","SEX","AGE","MARSTAT"),
  numVars = c("INCOME","ASSETS","DEBTS")
)
sdc <- varToFactor(sdc, "REGION")
sdc <- varToFactor(sdc, "SEX")
sdc <- varToFactor(sdc, "AGE")
sdc <- varToFactor(sdc, "MARSTAT")
```

1. Create an sdc object with the free1 dataset using REGION, SEX, AGE and MARSTAT as categorical variables and INCOME, ASSETS and DEBTS as numeric variables.

```
# mdav, single, onedims
print("Mdav algorithm time:")
```

2. Execute the different microaggregation algorithms using K = 4 and compute the execution time for each of them. (K is the minimum size of each group).

```
## [1] "Mdav algorithm time:"
```

```
startMdav <- Sys.time()
mdavSdc <- microaggregation(obj = sdc, method = "mdav", aggr  = 4)
endMdav <- Sys.time()
mdav_time <- round(endMdav-startMdav,4)
mdav_time
```

```
## Time difference of 0.0658 secs
```

```
print("Single algorithm time:")
```

```
## [1] "Single algorithm time:"
```

```
startSingle <- Sys.time()
singleSdc <- microaggregation(obj = sdc, method = "single", aggr = 4)
endSingle <- Sys.time()
single_time <- round(endSingle-startSingle,4)
single_time
```

```
## Time difference of 0.1523 secs
```

```
print("Onedims algorithm time:")
```

```
## [1] "Onedims algorithm time:"
```

```
startOnedims <- Sys.time()
onedimsSdc <- microaggregation(obj = sdc, method = "onedims", aggr = 4)
endOnedims <- Sys.time()
onedims_time <- round(endOnedims-startOnedims,4)
onedims_time
```

```
## Time difference of 0.0527 secs
```

```r
mdav_risk <- mdavSdc@risk$numeric
single_risk <- singleSdc@risk$numeric
onedims_risk <- onedimsSdc@risk$numeric

mdav_util <- mdavSdc@utility$il1
single_util <- singleSdc@utility$il1
onedims_util <- onedimsSdc@utility$il1

mdav_eigen <- mdavSdc@utility$eigen
single_eigen <- singleSdc@utility$eigen
onedims_eigen <- onedimsSdc@utility$eigen
```

3. From the new_sdc object get the following risk and utility measurements. We can now create the dataframe with the results obtained.

```r
algorithm <- c("MDAV", "Single","Onedims")
time <- c(mdav_time, single_time, onedims_time)
risk <- c(mdav_risk, single_risk, onedims_risk)
utility <- c(mdav_util, single_util, onedims_util)
eigen <- c(mdav_eigen, single_eigen, onedims_eigen)

df_results <- data.frame(algorithm, time, risk, utility, eigen)

print(df_results)
```

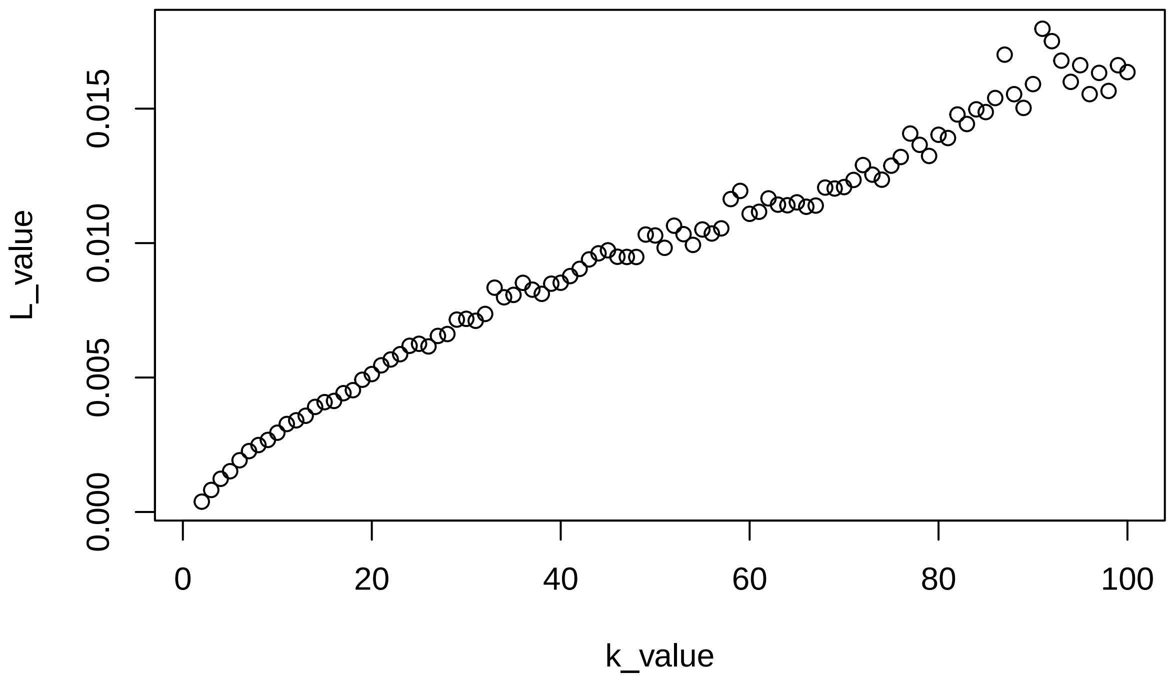
```
##   algorithm        time    risk     utility        eigen
## 1      MDAV 0.0658 secs 0.05875  54903.6037 0.0001568712
## 2    Single 0.1523 secs 0.00000 431155.9111 0.0269553104
## 3   Onedims 0.0527 secs 1.00000    770.4681 0.0001238770
```

From the results presented above, we can justify why MDAV is the most widely used microaggregation algorithm since it achieves a good balance between risk and utility. It has a lower risk value, indicating that it preserves more of the original data, and a lower utility, indicating that it distorts the data less. Regarding the eigen column measuring the loss of correlation between the variables and MDAV also performs well. Additionally, MDAV also has a reasonable running time, resulting a good choice when choosing a microaggregation algorithm.

b)In order to check the trade-off between risk and utility, use the sdc object created in the previous section and run mdav for values of K from 2 to 100. Calculate the L parameter in each case. Plot K vs L.

```r
histogram = c()
```

```r
original <- sdc@origData[,sdc@numVars]
meanOrig <- c(mean(original[,1]), mean(original[,2]), mean(original[,3]))
sst <- sum((original - meanOrig)^2)
for (k in 2:100) {
  sdcMdavFor <- microaggregation(obj = sdc, method = "mdav", aggr = k)
  manip <- sdcMdavFor@manipNumVars
  sse <- sum((original - manip)^2)
  L <- sse / sst
  histogram[k] <- L
}
plot(histogram, xlab = "k_value", ylab = "L_value")
```



Finally we can observe the correlation between the K and L value obtained when increasing the value of K.