Choosing a starting column in NIPALS

Kevin Wright

October 23, 2018

When the NIPALS algorithm is applied to a matrix X, it is necessary to choose a column of X as an initial vector for the iterations to calculate each principal component.

There appears to be no published research on how this column should be chosen.

Based on a limited investigation, we recommend chosing the column of X that has the largest sum of absolute values.

Methodology

In the code below, we first define some candidate functions for choosing the column of X with the maximum value of said functions. Using a small matrix of data, 10% of the values are set to NA. The NIPALS algorithm is applied with each of the candidate functions. The total number of iterations used to calculate the principal components is returned.

```
library(nipals)
corn <- structure(c(20.73, 23.58, 22.41, 19.97, 21.42, 24.48, 23.19, 25.73,
                    22.66, 23.93, 18.79, 18.56, 18.47, 18.33, 20.01, 19.03,
                    19.36, 21.2, 19.17, 18.17, 20.41, 17.67, 17.89, 21.41,
                    18.81, 22.77, NA, 19.86, 19.43, NA, 17.28, 14.9, 16.52,
                    14.77, 19.35, 22.46, 24.61, NA, NA, 26.16, NA, 19.48,
                    NA, 20.72, 17.8, 18.55, 19.56, 20.01, 20.05, 17.18,
                    16.29, 17.41, 15.86, 15.7, 17.84, 24.1, 27.02, 26.76,
                    26, 26.02, 20.63, 20.37, 21.17, 21.55, 19.12),
                  .Dim = c(5L, 13L),
                  .Dimnames = list(c("G1", "G2", "G3", "G4", "G5"),
                                   c("E01", "E02", "E03", "E04", "E05",
                                     "E06", "E07", "E08", "E09", "E10",
                                     "E11", "E12", "E13")))

# variance of values
myvar <- function(x) var(x, na.rm=TRUE)
# most extreme value
maxabs <- function(x) max(abs(x), na.rm=TRUE)
# range of values
rng <- function(x) diff(range(x, na.rm=TRUE))
# MAD
mymad <- function(x) mad(x, na.rm=TRUE)
# sum of absolute values
sumabs <- function(x) sum(abs(x), na.rm=TRUE)
# mean absolute value
meanabs<- function(x) mean(abs(x), na.rm=TRUE)
# variance of values, shrunk by fraction of complete values
shrinkvar <- function(x) var(x, na.rm=TRUE) * length(na.omit(x)) / length(x)

set.seed(42)
```

```
nrun <- 100
out <- data.frame(myvar=rep(NA, nrun), maxabs=rep(NA, nrun),
                  rng=rep(NA, nrun), mymad=rep(NA, nrun),
                  sumabs=rep(NA, nrun),meanabs=rep(NA, nrun),
                  shrinkvar=rep(NA, nrun))
for(kk in 1:nrun) {
  cat("kk=",kk, "\n")
  mat <- corn; probmiss = .1
  #mat <- as.matrix(uscrime); probmiss=.5
  #mat <- as.matrix(auto[,-1]); probmiss=.2
  mat[matrix(rbinom(prod(dim(mat)), size=1, prob=probmiss), nrow=nrow(mat)) > 0] <- NA
  tryCatch( {out$myvar[kk] <- sum(nipals(mat, verbose=TRUE, startcol=myvar)$iter) },
           error=function(e){} )
  tryCatch( {out$maxabs[kk] <- sum(nipals(mat, verbose=TRUE, startcol=maxabs)$iter) },
           error=function(e){} )
  tryCatch( {out$rng[kk] <- sum(nipals(mat, verbose=TRUE, startcol=rng)$iter) },
           error=function(e){} )
  tryCatch( {out$mymad[kk] <- sum(nipals(mat, verbose=TRUE, startcol=mymad)$iter) },
           error=function(e){} )
  tryCatch( {out$sumabs[kk] <- sum(nipals(mat, verbose=TRUE, startcol=sumabs)$iter) },
                error=function(e){} )
  tryCatch( {out$meanabs[kk] <- sum(nipals(mat, verbose=TRUE, startcol=meanabs)$iter) },
           error=function(e){} )
  tryCatch( {out$shrinkvar[kk] <- sum(nipals(mat, verbose=TRUE, startcol=shrinkvar)$iter)
           error=function(e){} )
}
```

The out dataframe looks something like this:

```
head(out)
#     myvar maxabs rng mymad sumabs meanabs shrinkvar
# 1     526     55  65    46     61     528       526
# 2     135     67  66    73     70     119       540
# 3     524     38  42    41     45      46       524
# 4      45     34  42    43     45      45        45
# 5     128    105 106   135    107     107       105
# 6     532     94  94    91    531     529        91
# 7     526     43  66    NA    526     527       526
# 8      74     64  74    60     63      63       554
# 9      58     52 538    54     57      57       538
# 10     NA     10  10    NA     10      NA        NA
```

We take the view that if the algorithm did not converge in less 500 iterations, it really did did not converge at all, so we replace values larger than 500 with NA, and then calculate the number of times that the algorithm converged and the total number of iterations used.

```
out <- as.data.frame(lapply(out, function(x) ifelse(x > 500, NA, x)))
library(purrr)
map_dbl(out, ~ sum(!is.na(.x))) %>% sort
#    myvar shrinkvar   meanabs     mymad       rng    sumabs    maxabs
#       51        55        75        76        85        86        92
map_dbl(out, ~ median(.x, na.rm=TRUE)) %>% sort
#      rng     mymad    maxabs    sumabs   meanabs shrinkvar     myvar
#     51.0      52.0      52.5      52.5      54.0      59.0      62.0
```

```
#plot(out)
```

The maxabs function converges 92/100 times for this data. For the remaining 8 times, sometimes one of the other functions allowed for convergence.

```
library(dplyr)
filter(out, is.na(maxabs))
#   myvar maxabs rng mymad sumabs meanabs shrinkvar
# 1    NA     NA  NA    NA     53      53        66
# 2    NA     NA  NA   160    168     158        NA
# 3    NA     NA  NA    NA     50      NA        NA
# 4    NA     NA  NA    NA     NA      NA        NA
# 5    NA     NA  NA    NA     NA      NA        NA
# 6    NA     NA  NA    NA     NA      NA        NA
# 7    NA     NA  NA    NA     NA      NA        83
# 8    NA     NA  53    63     58      59        NA
```

Results

We used 3 different datasets in the `tests` folder of the `nipals` package.

Using the `corn` data with 10% missing:

```
map_dbl(out, ~ sum(!is.na(.x))) %>% sort
#    myvar shrinkvar   meanabs     mymad    maxabs    sumabs       rng
#       60        61        75        81        87        90        96
map_dbl(out, ~ median(.x, na.rm=TRUE)) %>% sort
#   maxabs       rng    sumabs     mymad   meanabs     myvar shrinkvar
#     52.0      52.0      52.5      54.0      54.0      57.5      59.0
```

Using the `uscrime` data with 50% missing:

```
map_dbl(out, ~ sum(!is.na(.x))) %>% sort
#  meanabs     myvar    maxabs       rng     mymad shrinkvar    sumabs
#       55        57        57        57        57        60        62
map_dbl(out, ~ median(.x, na.rm=TRUE)) %>% sort
#    myvar shrinkvar    sumabs       rng     mymad   meanabs    maxabs
#    123.0     123.0     126.5     129.0     129.0     129.0     134.0
```

Using the `auto` data with 20% missing

```
map_dbl(out, ~ sum(!is.na(.x))) %>% sort
#    myvar    maxabs shrinkvar     mymad   meanabs    sumabs       rng
#       38        40        41        45        48        61        64
map_dbl(out, ~ median(.x, na.rm=TRUE)) %>% sort
#   sumabs       rng shrinkvar     mymad    maxabs     myvar   meanabs
#     54.0      54.5      61.0      63.0      64.0      64.5      66.5
```

Conclusion

The function `myvar` (variance) is consistently poor. This was the default method used up to and through `nipals` version 0.3.

The function sumabs (sum of absolute values) is consistently good, with relatively high rates of convergence and low total number of iterations.

Based on this limited research, it was decided to use the column of X that had the greatest sum of absolute values. This does have some intuitive appeal. Since the NIPALS algorithm has some least-squares type of calculations, extreme values might have the most influence in the algorithm.