The Factor Forest: Step-by-Step

```
library(mlr)
library(psych)
library(ineq)
library(BBmisc)
library(ddpcr)
```

```
set.seed(1234) # set seed for reproducibility
```

```
# source scripts for helper functions

source("xgb-functions.R")
source("CD_approach.R")

# load pre-trained model from Goretzko and Bühner (2020)

xgb <- readRDS(file = "tunedxgb.rds")
```

Simulate Exemplary Data

For this step-by-step example of how the Factor Forest works, we first simulate data based on a two-factor model with five manifest indicators per latent variable. The true standardized loading matrix (i.e., the loading pattern on population level) for our simulation is

$$\mathbf{\Lambda} = \begin{bmatrix} 0.7 & 0.2 \\ 0.7 & 0.2 \\ 0.7 & 0.2 \\ 0.7 & 0.2 \\ 0.7 & 0.2 \\ 0.2 & 0.7 \\ 0.2 & 0.7 \\ 0.2 & 0.7 \\ 0.2 & 0.7 \\ 0.2 & 0.7 \end{bmatrix}$$

and no further inter-factor correlations or correlated residuals are assumed. Based on this loading matrix, the true manifest correlation matrix on population level is $\mathbf{\Sigma} = \mathbf{\Lambda}\mathbf{\Lambda}^\top + \mathbb{1}_{p\times p} - diag(\mathbf{\Lambda}\mathbf{\Lambda}^\top)$. Accordingly, the population correlation matrix is

```
L = matrix(c(rep(0.7,5), rep(0.2,10), rep(0.7,5)), ncol = 2)
Sigma = L%*%t(L) + diag(diag(diag(10)-L%*%t(L)))
as.data.frame(Sigma, row.names = paste("V", 1:10, sep=""),
              col.names = paste("V", 1:10, sep=""))
```

```
##       V1   V2   V3   V4   V5   V6   V7   V8   V9  V10
## V1  1.00 0.53 0.53 0.53 0.53 0.28 0.28 0.28 0.28 0.28
## V2  0.53 1.00 0.53 0.53 0.53 0.28 0.28 0.28 0.28 0.28
## V3  0.53 0.53 1.00 0.53 0.53 0.28 0.28 0.28 0.28 0.28
## V4  0.53 0.53 0.53 1.00 0.53 0.28 0.28 0.28 0.28 0.28
## V5  0.53 0.53 0.53 0.53 1.00 0.28 0.28 0.28 0.28 0.28
## V6  0.28 0.28 0.28 0.28 0.28 1.00 0.53 0.53 0.53 0.53
## V7  0.28 0.28 0.28 0.28 0.28 0.53 1.00 0.53 0.53 0.53
## V8  0.28 0.28 0.28 0.28 0.28 0.53 0.53 1.00 0.53 0.53
## V9  0.28 0.28 0.28 0.28 0.28 0.53 0.53 0.53 1.00 0.53
## V10 0.28 0.28 0.28 0.28 0.28 0.53 0.53 0.53 0.53 1.00
```

which we use to draw a sample of 500 observations assuming multivariate normality and then using the thresholds $-1.5, -0.5, 0.5, 1.5$ as cut-points to discretize the data reflecting answers on a five-point Likert scale and underlying multivariate normal data.

```
N = 500
data = data.frame(mvtnorm::rmvnorm(N, mean = rep(0,10), sigma = Sigma))
data = data.frame(lapply(data,
                           FUN = function(x){cut(x, c(min(x)-0.01,
                                                    c(-1.5, -0.5, 0.5, 1.5),
                                                    max(x) + 0.01), labels = FALSE,
                                               right =FALSE)}))
```

Calculate Features that were used to Train the XGBoost Model

Goretzko and Bühner (2020) trained the *xgboost* model on simulated data sets for which 184 features were extracted. These features are used as predictor variables in the pre-trained model. Hence, the Factor Forest predicts the number of factors to retain based on the respective values of these features.

To apply the pre-trained model, we have to calculate all 184 features. The majority of these features is based on the empirical correlation matrix of the manifest variables. Here, in our data sample, the ten five-point Likert items should be analyzed. In a first step, we therefore calculate the correlation matrix using polychoric correlations to address the ordinal character of the manifest variables.

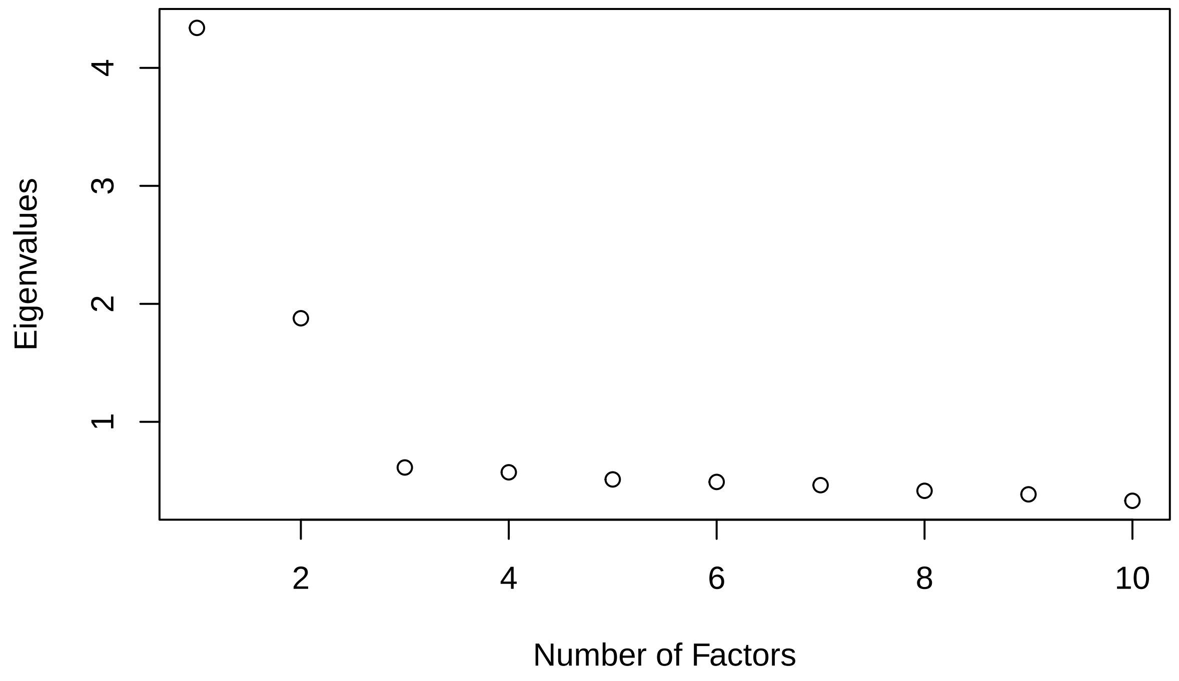
```
dat_cor = polychoric(data)$rho

as.data.frame(dat_cor, row.names = paste("V", 1:10, sep=""),
              col.names = paste("V", 1:10, sep=""))
```

```
##            X1        X2        X3        X4        X5        X6        X7
## V1  1.0000000 0.5400762 0.5488181 0.5424075 0.4984558 0.1561632 0.1968022
## V2  0.5400762 1.0000000 0.5466025 0.5330946 0.5629533 0.3115140 0.2647598
## V3  0.5488181 0.5466025 1.0000000 0.4486184 0.5257562 0.2673469 0.2551763
## V4  0.5424075 0.5330946 0.4486184 1.0000000 0.5577139 0.2621238 0.2411450
## V5  0.4984558 0.5629533 0.5257562 0.5577139 1.0000000 0.3143640 0.2785985
## V6  0.1561632 0.3115140 0.2673469 0.2621238 0.3143640 1.0000000 0.4714896
## V7  0.1968022 0.2647598 0.2551763 0.2411450 0.2785985 0.4714896 1.0000000
## V8  0.2281808 0.3646888 0.2160764 0.2753082 0.3455044 0.5387075 0.5084473
## V9  0.1890146 0.2618931 0.1637554 0.1962743 0.1879813 0.4865643 0.5321920
## V10 0.1484129 0.2803855 0.2491795 0.2644078 0.2479185 0.5470089 0.5111764
##            X8        X9       X10
## V1  0.2281808 0.1890146 0.1484129
## V2  0.3646888 0.2618931 0.2803855
## V3  0.2160764 0.1637554 0.2491795
## V4  0.2753082 0.1962743 0.2644078
## V5  0.3455044 0.1879813 0.2479185
## V6  0.5387075 0.4865643 0.5470089
## V7  0.5084473 0.5321920 0.5111764
## V8  1.0000000 0.5203105 0.5621649
## V9  0.5203105 1.0000000 0.4935226
## V10 0.5621649 0.4935226 1.0000000
```

The eigenvalues of this matrix are central features for factor retention as most of the common factor retention criteria (parallel analysis, Kaiser criterion, Scree-test, etc.) rely on their empirical distribution.

```
eigval = eigen(dat_cor)$values
plot(1:10, eigval, ylab = "Eigenvalues", xlab = "Number of Factors")
```



These eigenvalues are directly used as input variables or features for the Factor Forest model, and they are used to calculate further features considering the proportion of explained variance associated with an eigenvalue, the heterogeneity of the eigenvalues as well as common criteria such as the eigenvalue-greater-one rule or the empirical Kaiser criterion:

```
p = nrow(dat_cor) # set the number of manifest variables
vareig = cumsum(eigval)/p

# calculate eigenvalue-based features

eiggreater1 = sum(eigval > 1)  # Kaiser criterion
```

```
lref = rep(0,p)
for (i in 1:p) {
  lref[i] = max(((1 + sqrt(p/N))^2) * (p-sum(lref))/(p-i+1),1)
}
ekc = which(eigval<=lref)[1]-1 # empirical Kaiser criteiron
releig1 = eigval[1]/p # relative size of the first eigenvalue
releig2 = sum(eigval[1:2])/p # relative size of the first two eigenvalues
releig3 = sum(eigval[1:3])/p # relative size of the first three eigenvalues
eiggreater07 = sum(eigval > 0.7) # number of eigenvalues > 0.7
sdeigval = sd(eigval) # SD as a measure of heterogeneity of the eigenvalues

# number of eigenvalues that correspond to 50%/75% explained variance
var50 = min(which(vareig > 0.50))
var75 = min(which(vareig > 0.75))
```

As the first two eigenvalues are comparably large (4.339 and 1.878), they are associated with approx. 62.2% of the indicators' variance. Both the Kaiser criterion and the empirical Kaiser criterion suggest retaining two factors.

```
ekc
```

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

```
eiggreater1
```

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

Besides eigenvalue-based features, the Factor Forest also relies on matrix norms that can be used to describe and quantify the correlation matrix as a whole.

```
onenorm = norm(dat_cor,"O") # L1-norm or maximum absolute column sum
frobnorm = norm(dat_cor,"F") # Frobenius norm (L2) or Euclidean norm
maxnorm = norm(dat_cor-diag(p),"M")
# Maximum norm of cor-matrix without diagonal (maximum of all bivariate correlations)
avgcor = sum(abs(dat_cor-diag(p)))/(p*(p-1))
# average of absolute values of bivariate correlations
specnorm = sqrt(eigen(t(dat_cor)%*%dat_cor)$values[1]) # spectral norm
```

Furthermore, the number of bi-variate correlations whose absolute value is equal to or smaller than 0.1, the average squared multiple correlation (which is the average communality estimate) as well as the determinant of the correlation matrix are calculated as features.

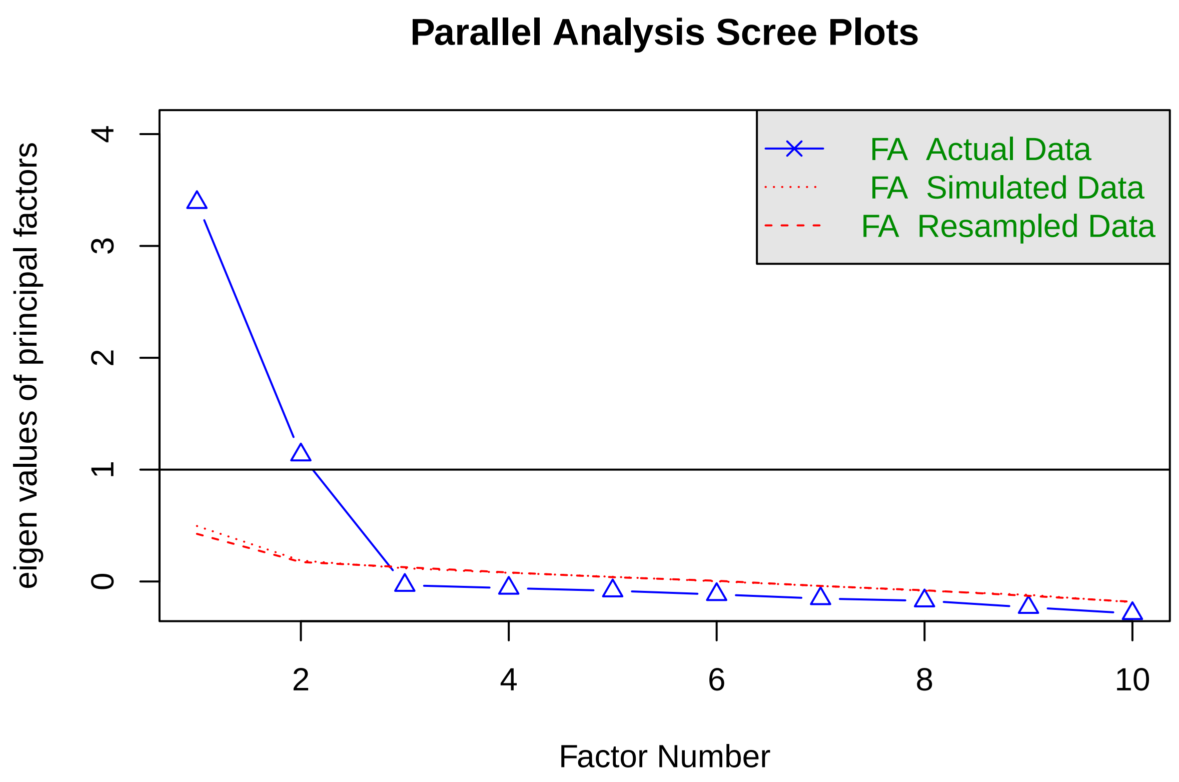
```
smlcor = sum(abs(dat_cor) <= 0.1) # number of "small" correlations
avgcom = mean(smc(dat_cor)) # average communality estimate
det = det(dat_cor) # determinant of correlation matrix
```

Based on the correlation matrix some "inequality" measures are calculated to quantify the homo- or heterogeneity of the bi-variate correlations.

```
KMO = KMO(dat_cor)$MSA # Measure of sampling adequacy known from Kaiser-Meyer-Olkin test
Gini = ineq(abs(lower.tri(dat_cor)), type = "Gini") # Gini coefficient
Kolm = ineq(abs(lower.tri(dat_cor)), type = "Kolm") # Kolm measure
```

A parallel analysis is also conducted and the suggested number of factors as well as the reduced eigenvalues that are based on the communalities implied by the common factor model (not the full correlation matrix) are used as features to predict the number of factors.

```
pa = fa.parallel(data, fa="fa",plot = TRUE)
```



```
## Parallel analysis suggests that the number of factors =  2  and the number of components =  NA
```

```
pa_solution = pa$nfact
fa_eigval = pa$fa.values

data.frame(Factors = 1:10, Eigenvalues = eigval, Reduced_Eigenvalues = fa_eigval)
```

```
##    Factors Eigenvalues Reduced_Eigenvalues
## 1        1   4.3388988          3.38924692
## 2        2   1.8779546          1.13261089
## 3        3   0.6130353         -0.03359873
## 4        4   0.5723390         -0.05922389
## 5        5   0.5118498         -0.08269306
## 6        6   0.4907466         -0.11594711
## 7        7   0.4631880         -0.15249682
## 8        8   0.4154288         -0.17231288
## 9        9   0.3856388         -0.23066779
## 10      10   0.3309203         -0.28567009
```

Parallel analysis and the related comparison data approach also agree on two factors.

```
cd = EFA.Comp.Data(Data = data, F.Max = 8, use = "pairwise.complete.obs")  # Comparison Data Approach

pa_solution
```

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

```
cd
```

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

Since the pre-trained *xgboost* model was trained on data with 80 indicators at maximum, the vectors containing the empirical eigenvalues (or reduced eigenvalues) are artificially extended (the model takes 80 eigenvalues as input - all “missing” eigenvalues are coded as -1000).

```
eigval[(length(eigval)+1):80] = -1000
fa_eigval[(length(fa_eigval)+1):80] = -1000
# standardize naming of eigenvalue vectors
names(eigval) <- paste("eigval", 1:80, sep = "")
names(fa_eigval) <- paste("fa_eigval", 1:80, sep = "")
```

Predict the Number of Factors based on the Calculated Features

All calculated features are now combined to one data set that is passed to the prediction function of the *xgboost* model. Based on these values the pre-trained model then estimates the number of factors that most likely underlies the empirical data.

```
# combine features to data frame (input for trained model)

features = cbind(data.frame(N,p,eiggreater1, releig1, releig2, releig3, eiggreater07,
                             sdeigval, var50, var75, onenorm, frobnorm, maxnorm,
                             avgcor, specnorm, smlcor, avgcom, det, KMO, Gini, Kolm,
                             pa_solution, ekc, cd), t(eigval), t(fa_eigval))

# the xgboost model predicts the number of factors given the calculated features

out = predict(xgb, newdata = features)$data
```

```
## [18:02:27] WARNING: amalgamation/../src/learner.cc:851: Loading model from XGBoost < 1.0.0, consider
```

```
cat("\n\n", "Probabilities for different factor solutions:", "\n\n")
```

```
##
##
##  Probabilities for different factor solutions:
```

```
print(round(out[-9], digits = 5))
```

```
##   prob.1  prob.2 prob.3 prob.4 prob.5 prob.6 prob.7 prob.8
## 1      0 0.99999  1e-05      0      0      0      0      0
```

```
cat("\n\n", paste("Suggested number of factors:", out$response, sep =" "), "\n\n")
```

```
##
##
##  Suggested number of factors: 2
```

The pre-trained *xgboost* model also suggests two factors (the presented probabilites show that the model is very “certain” that two factors is the correct solution).