Supplementary source code

This section provides

1. Source code for the simulation of outlying protein levels for individuals with average genotypes (low-leverage observations) including the calculation of the statistical parameters
2. Source code for the real data applications

Note: The working directories have to be set accordingly in each R script. Be aware of time consuming computations.

Simulation study

Source code for the simulation of outlying protein levels for individuals with average genotypes (low-leverage observations) comprising the use of standard and robust Huber-LASSO and the calculation of the Jaccard index, false-positive rate, true-positive rate and estimated regression coefficients

Structure:

1. Data preparation
2. Standard and robust Huber-LASSO
3. Calculation of the statistical parameters
    a. Jaccard index
    b. False-positive rate
    c. True-positive rate
    d. Estimated regression coefficients
4. Diagnostic plots

```
# ------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------ #
# 1. Data preparation
# ------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------ #

#Load packages
library(hqreg)
library(depth)

#Working directory
dir.read <- "Directoy to read data"
dir.res <- "Directory to save results"

#Read genotype data####
setwd(dir.read)
#Genotype data of associated SNPs
X_geno_ass <- read.table("dat_ass.txt", header = T)
#Genotype data of non-associated SNPs
X_geno_add <- read.table("dat_nonass.txt", header = T)
#Combine genotype data
identical(X_geno_ass[,1],X_geno_add[,1])#compare Sample_Names
X <- as.matrix(cbind(X_geno_ass, X_geno_add))
X <- X[,-grep("Sample_Name",colnames(X))]

#Read proteomics data
Proteomics.data <- read.delim("Proteomics data",header = TRUE)

#Compute trivariate depth of individuals
forpca <- Proteomics.data [, c("Sample_Name", "PC1", "PC2", "PC3")]
for (i in 1:length(forpca$Sample_Name)) {
  forpca$depth[i] <- depth(forpca[i, 2:4], forpca[, 2:4], approx = TRUE)
}

#Read DEFB119 values
identical(Proteomics.data$Sample_Name, X[,1])
y <- Proteomics.data$DEFB119.13455.10.3

#Number of iterations
niter <- 100

#Number of individuals per iteration
nind <- 500

#Simulated outlier values
outlier <- c(-5, -2.5, 0, 2.5, 5)

#Define variables for r and r^2 for standard and robust Huber-LASSO
r_standard <- matrix(nrow = niter, ncol = length(outlier))
r2_standard <- matrix(nrow = niter, ncol = length(outlier))
```

```
r_robust <- matrix(nrow = niter, ncol = length(outlier))
r2_robust <- matrix(nrow = niter, ncol = length(outlier))


# ------------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------------ #
# 2. LASSO and 5-fold outcome prediction
# ------------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------------ #

for (i in 1:niter) {
  print(paste0("Iteration number ",i," out of ",niter))
  #Set seed
  set.seed(i)
  #Draw samples (for subset definition)
  rd_smp <- sample(1:nrow(X), nind)

  #Define subsets
  fold1 <- rd_smp[1:(round(0.2 * nind, 0))]
  fold2 <- rd_smp[(round(0.2 * nind, 0) + 1):(round(0.4 * nind, 0))]
  fold3 <- rd_smp[(round(0.4 * nind, 0) + 1):(round(0.6 * nind, 0))]
  fold4 <- rd_smp[(round(0.6 * nind, 0) + 1):(round(0.8 * nind, 0))]
  fold5 <- rd_smp[(round(0.8 * nind, 0) + 1):nind]

  for (j in 1:length(outlier)) {
    #LASSO####
    #Select low-leverage observation by highest trivariate depth
    #Note: replace min() by max() to select high-leverage observation
    idx <- which(forpca_rd$depth == min(forpca_rd$depth))[1]
    #Define simulated outlying protein residual
    y_sim <- y[rd_smp]
    y_sim[idx] <- outlier[j]
    X_sim <- X[rd_smp,]

    #Model trained on all individuals
    cv_standard <- cv.hqreg(X_sim,y_sim,method="ls",FUN="hqreg_raw")
    cv_robust <- cv.hqreg(X_sim,y_sim,method="huber",FUN="hqreg_raw")

    #Standard LASSO results
    #Regression parameter
    lambda_standard[(j-1)*niter+i] <- cv_standard$lambda.min
    #Selected SNPs
    snps_standard[
      (j-1)*niter+i, 1:(predict(cv_standard, lambda=cv_standard$lambda.min, type="nvars")+1)
      ] <- names(coef(cv_standard, cv_standard$lambda.min)[
                coef(cv_standard,cv_standard$lambda.min)!=0])
    #Regression coefficient estimates of selected SNPs
    coef_standard[
      (j-1)*niter+i, 1:(predict(cv_standard, lambda=cv_standard$lambda.min, type="nvars")+1)
      ] <- coef(cv_standard, cv_standard$lambda.min)[
                coef(cv_standard, cv_standard$lambda.min)!=0]

    #Robust Huber-LASSO results
    #Regression parameter
    lambda_robust[(j-1)*niter+i] <- cv_robust$lambda.min
    #Selected SNPs
    snps_robust[
      (j-1)*niter+i, 1:(predict(cv_robust, lambda=cv_robust$lambda.min, type="nvars")+1)
      ] <- names(coef(cv_robust, cv_robust$lambda.min)[
                coef(cv_robust, cv_robust$lambda.min)!=0])
    #Regression coefficient estimates of selected SNPs
    coef_robust[
      (j-1)*niter+i, 1:(predict(cv_robust, lambda=cv_robust$lambda.min, type="nvars")+1)
      ] <- coef(cv_robust, cv_robust$lambda.min)[coef(cv_robust, cv_robust$lambda.min)!=0]


    #5-fold prediction####
    #Select low-leverage observation in Fold2-5
    #Note: replace min() by max() to define high-leverage observation
    idx <- which(which(forpca$depth == min(forpca$depth[c(fold2, fold3, fold4, fold5)])) %in%
                   c(fold2, fold3, fold4, fold5))[1]
    #Define simulated outlying protein residual
    y_sim <- y
    y_sim[idx] <- outlier[j]

    #Model trained on Fold2-5
    cv_standard <-
      cv.hqreg(X[-fold1, ], y_sim[-fold1], method = "ls", FUN = "hqreg_raw", seed = 111)
    cv_robust <-
      cv.hqreg(X[-fold1, ], y_sim[-fold1], method = "huber", FUN = "hqreg_raw", seed = 111)


    #Plasma protein residuals prediction for Fold1
    pred_standard[1:(round(0.2 * nind, 0))] <-
      predict(cv_standard, X[fold1, ], lambda = "lambda.min")
    pred_robust[1:(round(0.2 * nind, 0))] <-
      predict(cv_robust, X[fold1, ], lambda = "lambda.min")

    #Select low-leverage observation in Fold1 + Fold3-5
    #Note: replace min() by max() to define high-leverage observation
    idx <- which(which(forpca$depth == min(forpca$depth[c(fold1, fold3, fold4, fold5)])) %in%
                   c(fold1, fold3, fold4, fold5))[1]
    #Define simulated outlying protein residual
    y_sim <- y
    y_sim[idx] <- outlier[j]
```

```
    #Model trained on Fold1 + Fold3-5
    cv_standard <-
      cv.hqreg(X[-fold2, ], y_sim[-fold2], method = "ls", FUN = "hqreg_raw", seed = 111)
    cv_robust <-
      cv.hqreg(X[-fold2, ], y_sim[-fold2], method = "huber", FUN = "hqreg_raw", seed = 111)

    #Predict plasma protein residual for Fold2
    pred_standard[(round(0.2 * nind, 0) + 1):(round(0.4 * nind, 0))] <-
      predict(cv_standard, X[fold2, ], lambda = "lambda.min")
    pred_robust[(round(0.2 * nind, 0) + 1):(round(0.4 * nind, 0))] <-
      predict(cv_robust, X[fold2, ], lambda = "lambda.min")

    #Select low-leverage observation in Fold1-2 + Fold4-5
    #Note: replace min() by max() to define high-leverage observation
    idx <- which(which(forpca$depth == min(forpca$depth[c(fold1, fold2, fold4, fold5)])) %in%
                   c(fold1, fold2, fold4, fold5))[1]
    #Define simulated outlying protein residual
    y_sim <- y
    y_sim[idx] <- outlier[j]

    #Model trained on Fold1-2 + Fold4-5
    cv_standard <-
      cv.hqreg(X[-fold3, ], y_sim[-fold3], method = "ls", FUN = "hqreg_raw", seed = 111)
    cv_robust <-
      cv.hqreg(X[-fold3, ], y_sim[-fold3], method = "huber", FUN = "hqreg_raw", seed = 111)

    #Predict plasma protein residual for Fold3
    pred_standard[(round(0.4 * nind, 0) + 1):(round(0.6 * nind, 0))] <-
      predict(cv_standard, X[fold3, ], lambda = "lambda.min")
    pred_robust[(round(0.4 * nind, 0) + 1):(round(0.6 * nind, 0))] <-
      predict(cv_robust, X[fold3, ], lambda = "lambda.min")

    #Select low-leverage observation in Fold1-3 + Fold5
    #Note: replace min() by max() to define high-leverage observation
    idx <- which(which(forpca$depth == min(forpca$depth[c(fold1, fold2, fold3, fold5)])) %in%
                   c(fold1, fold2, fold3, fold5))[1]
    #Define simulated outlying protein residual
    y_sim <- y
    y_sim[idx] <- outlier[j]

    #Model trained on Fold1-3 + Fold5
    cv_standard <-
      cv.hqreg(X[-fold4, ], y_sim[-fold4], method = "ls", FUN = "hqreg_raw", seed = 111)
    cv_robust <-
      cv.hqreg(X[-fold4, ], y_sim[-fold4], method = "huber", FUN = "hqreg_raw", seed = 111)

    #predict plasma protein residual for Fold4
    pred_standard[(round(0.6 * nind, 0) + 1):(round(0.8 * nind, 0))] <-
      predict(cv_standard, X[fold4, ], lambda = "lambda.min")
    pred_robust[(round(0.6 * nind, 0) + 1):(round(0.8 * nind, 0))] <-
      predict(cv_robust, X[fold4, ], lambda = "lambda.min")

    #Select low-leverage observation in Fold1-4
    #Note: replace min() by max() to define high-leverage observation
    idx <- which(which(forpca$depth == min(forpca$depth[c(fold1, fold2, fold3, fold4)])) %in%
                   c(fold1, fold2, fold3, fold4))[1]
    #Define simulated outlying protein residual
    y_sim <- y
    y_sim[idx] <- outlier[j]

    #Model trained on Fold1-4
    cv_standard <-
      cv.hqreg(X[-fold5, ], y_sim[-fold5], method = "ls", FUN = "hqreg_raw", seed = 111)
    cv_robust <-
      cv.hqreg(X[-fold5, ], y_sim[-fold5], method = "huber", FUN = "hqreg_raw", seed = 111)

    #Predict plasma protein residual for Fold5
    pred_standard[(round(0.8 * nind, 0) + 1):nind] <-
      predict(cv_standard, X[fold5, ], lambda = "lambda.min")
    pred_robust[(round(0.8 * nind, 0) + 1):nind] <-
      predict(cv_robust, X[fold5, ], lambda = "lambda.min")

    #Reorder protein residuals to calculate the correlation
    y_true <- y[rd_smp]

    #Correlation of observed and predicted plasma protein residuals
    r_standard[i, j] <- cor(y_true, pred_standard,method="kendall")
    r_robust[i, j] <- cor(y_true, pred_robust,method="kendall")

  }
}


# ------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------ #
# 3. Statistical parameters
# ------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------ #

#a. Jaccard index####
#Initiate variables for standard and robust Jaccard index
jaccard_standard_raw <- matrix(nrow = niter * length(outlier), ncol = (niter - 1))
jaccard_robust_raw <- matrix(nrow = niter * length(outlier), ncol = (niter - 1))

jaccard_standard <- matrix(nrow = (niter * (niter - 1)) / 2, ncol = length(outlier))
```

```
jaccard_robust <- matrix(nrow = (niter * (niter - 1)) / 2, ncol = length(outlier))

for (m in 1:length(outlier)) {
  for (i in ((1 * (m - 1) * niter) + 1):((niter * m) - 1)) {
    for (j in (i + 1):(niter * m)) {
      #Compute Jaccard index from standard LASSO
      jaccard_standard_raw[i, j - 1 - ((m - 1) * niter)] <-
        length(intersect(snps_standard[i, which(!(is.na(snps_standard[i, ])))[-1]],
                         snps_standard[j, which(!(is.na(snps_standard[j, ])))[-1]])) /
        length(union(snps_standard[i, which(!(is.na(snps_standard[i, ])))[-1]],
                     snps_standard[j, which(!(is.na(snps_standard[j, ])))[-1]]))
      if (is.na(jaccard_standard_raw[i, j - 1 - ((m - 1) * niter)])) {
        jaccard_standard_raw[i, j - 1 - ((m - 1) * niter)] <- 1
      }
      #Compute Jaccard index from robust Huber-LASSO
      jaccard_robust_raw[i, j - 1 - ((m - 1) * niter)] <-
        length(intersect(snps_robust[i, which(!(is.na(snps_robust[i, ])))[-1]],
                         snps_robust[j, which(!(is.na(snps_robust[j, ])))[-1]])) /
        length(union(snps_robust[i, which(!(is.na(snps_robust[i, ])))[-1]],
                     snps_robust[j, which(!(is.na(snps_robust[j, ])))[-1]]))
      if (is.na(jaccard_robust_raw[i, j - 1 - ((m - 1) * niter)])) {
        jaccard_robust_raw[i, j - 1 - ((m - 1) * niter)] <- 1
      }
    }
  }
  #Order Jaccard index by simulated outlying protein residual
  jaccard_standard[1:length(which(!(is.na(
    as.numeric(jaccard_standard_raw[(1 + (niter * (m - 1))):(niter * m),])))), m] <-
    as.numeric(jaccard_standard_raw[(1 + (niter * (m - 1))):(niter * m),])[which(!(is.na(
      as.numeric(jaccard_standard_raw[(1 + (niter * (m - 1))):(niter * m),])))]
  jaccard_robust[1:length(which(!(is.na(
    as.numeric(jaccard_robust_raw[(1 + (niter * (m - 1))):(niter * m), ])))), m] <-
    as.numeric(jaccard_robust_raw[(1 + (niter * (m - 1))):(niter * m), ])[which(!(is.na(
      as.numeric(jaccard_robust_raw[(1 + (niter * (m - 1))):(niter * m), ])))]
}


#b. False-positive rate####
#Non-associated SNP IDs
snps_nonass <- colnames(X_geno_add[,-grep("Sample_Name",colnames(X_geno_add))])

#Initiate variables for count of selected non-associated SNPs
count_standard <- matrix(nrow = niter, ncol = length(snps_nonass))
count_robust <- matrix(nrow = niter, ncol = length(snps_nonass))

#Initiate variables for standard and robust false-positive rate
type1_standard <- matrix(nrow = niter, ncol = length(outlier))
type1_robust <- matrix(nrow = niter, ncol = length(outlier))

for (m in 1:length(outlier)) {
  snps_standard_subset <- snps_standard[(1 + (niter * (m - 1))):(m * niter), ]
  coef_standard_subset <- coef_standard[(1 + (niter * (m - 1))):(m * niter), ]

  snps_robust_subset <- snps_robust[(1 + (niter * (m - 1))):(m * niter), ]
  coef_robust_subset <- coef_robust[(1 + (niter * (m - 1))):(m * niter), ]

  for (i in 1:niter) {
    for (j in 1:length(snps_nonass)) {
      count_standard[i, j] <-
        length(which(snps_standard_subset[i, ] %in% snps_nonass[j]))
      count_robust[i, j] <-
        length(which(snps_robust_subset[i, ] %in% snps_nonass[j]))
    }

    type1_standard[i, m] <- sum(count_standard[i, ]) / length(snps_nonass)
    type1_robust[i, m] <- sum(count_robust[i, ]) / length(snps_nonass)
  }
}

#c. True-positive rate####
#Associated SNP IDs
snps_ass <- colnames(X_geno_ass)[,-grep("Sample_Name",colnames(X_geno_ass))]

#Initiate variables for count of selected associated SNPs
count_standard <- numeric(niter)
count_robust <- numeric(niter)

#Initiate variables for standard and robust true-positive rate
power_standard <- matrix(nrow = length(snps_ass), ncol = length(outlier))
power_robust <- matrix(nrow = length(snps_ass), ncol = length(outlier))

for (i in 1:length(snps_ass)) {
  for (j in 1:length(outlier)) {
    #Subset selected SNPs and coefficient estimates by simulated outlying protein residual
    snps_standard_subset <- snps_standard[(1 + (niter * (j - 1))):(j * niter), ]
    coef_standard_subset <- coef_standard[(1 + (niter * (j - 1))):(j * niter), ]

    snps_robust_subset <- snps_robust[(1 + (niter * (j - 1))):(j * niter), ]
    coef_robust_subset <- coef_robust[(1 + (niter * (j - 1))):(j * niter), ]


    for (k in 1:niter) {
      #Check selection of associated SNPs in each iteration
      count_standard[k] <- length(which(snps_standard_subset[k, ] %in% snps_ass[i]))
      count_robust[k] <- length(which(snps_robust_subset[k, ] %in% snps_ass[i]))
    }
```

```
    #Compute true-positive rate
    power_standard[i, j] <- sum(count_standard) / niter
    power_robust[i, j] <- sum(count_robust) / niter
  }
}


#d. Estimated regression coefficients####
#Initiate variables for standard and robust regression coefficient estimates
coef_est_standard <-  matrix(nrow = length(snps_ass), ncol = length(outlier) * niter)
coef_est_robust <-  matrix(nrow = length(snps_ass), ncol = length(outlier) * niter)

#Regression coefficient estimates of associated SNP for each simulated outlying protein #residual and
iteration
for (j in 1:length(outlier)) {
  #Subset selected SNPs and coefficient estimates by simulated outlying protein residual
  snps_standard_subset <- snps_standard[(1 + (niter * (j - 1))):(j * niter), ]
  coef_standard_subset <- coef_standard[(1 + (niter * (j - 1))):(j * niter), ]

  snps_robust_subset <- snps_robust[(1 + (niter * (j - 1))):(j * niter), ]
  coef_robust_subset <- coef_robust[(1 + (niter * (j - 1))):(j * niter), ]

  for (i in 1:length(snps_ass)) {
    snps_idx_standard <- which(snps_standard_subset == snps_ass[i])
    if (length(snps_idx_standard) != 0) {
      coef_est_standard[i, (1 + (niter * (j - 1))):((niter * (j - 1)) +
        length(as.vector(coef_standard_subset)[snps_idx_standard]))] <-
        as.vector(coef_standard_subset)[snps_idx_standard]
    }

    snps_idx_robust <- which(snps_robust_subset == snps_ass[i])
    if (length(snps_idx_robust) != 0) {
      coef_est_robust[i, (1 + (niter * (j - 1))):((niter * (j - 1)) +
        length(as.vector(coef_robust_subset)[snps_idx_robust]))] <-
        as.vector(coef_robust_subset)[snps_idx_robust]
    }
  }
}
coef_est_standard[which(is.na(coef_est_standard))] <- 0
coef_est_robust[which(is.na(coef_est_robust))] <- 0

#e. Squared correlation###

#Fisher-consistent version of correlation
transf <- function(r){sin(0.5*pi*r)}
r_standard_transf <- matrix(nrow=nrow(r_standard),ncol=ncol(r_standard))
r_robust_transf <- matrix(nrow=nrow(r_robust),ncol=ncol(r_robust))
for (i in 1:nrow(r_standard)){
  for (j in 1:ncol(r_standard)){
    r_standard__transf[i,j] <- transf(r_standard[i,j])
    r_robust_transf[i,j] <- transf(r_robust[i,j])
  }}

r2_standard_transf<- matrix(ncol=ncol(r_standard_transf),nrow=nrow(r_standard_transf))
r2_robust_transf<- matrix(ncol=ncol(r_robust_transf),nrow=nrow(r_robust_transf))
for (i in 1:nrow(r_standard)){
  for (j in 1:ncol(r_standard)){
    r2_standard_transf[i,j] <- r_standard_transf[i,j]^2
    r2_robust_transf[i,j] <- r_robust_transf[i,j]^2
  }}



# ------------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------------ #
# 4. Diagnostic plots
# ------------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------------ #

#Set working directory
setwd(dir.res)

#Create plot as .tif with resolution equal to 600dpi

#Open graphic device
tiff("Figure2.tiff", height = 6, width = 5.5, units = 'in', compression="lzw", res=1200)


#Plot
boxplot(r2_standard_transf [,1],r2_robust_transf [,1],
        r2_standard_transf [,2],r2_robust_transf [,2],
        r2_standard_transf [,3],r2_robust_transf [,3],
        r2_standard_transf [,4],r2_robust_transf [,4],
        r2_standard_transf [,5],r2_robust_transf [,5],
        boxcol=rep(c("black","blue"),1), medcol=rep(c("black","blue"),1),
        whiskcol=rep(c("black","blue"),1), staplecol=rep(c("black","blue"),1),
        outcol=rep(c("black","blue"),1),
        at=c(1:2,4:5,7:8,10:11,13:14), xaxt="n",
        xlab="Simulated protein residual", ylab=""
)
axis(1,at=c(1.5,4.5,7.5,10.5,13.5),labels=c(-5,-2.5,0,2.5,5))
title(ylab="Squared correlation between observed and predicted protein residuals", line=2.5)

#Close device
dev.off()
```

Real data applications

Source code for the real data applications

Structure

1. Data preparation
2. Standard and robust Huber-LASSO
3. Diagnostic plot

```
# ------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------ #
# 1. Data preparation
# ------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------ #

#Load package
library(hqreg)

#Working directory
dir.read <- "Directoy to read data"
dir.res <- "Directory to save results"

#Read data####
setwd(dir.read)
dat <- read.table("dat.txt", header = T, stringsAsFactors = F)
#data containing associated and randomly sampled non-associated variants with outcome

#Genotypes
X <- as.matrix(dat[, grep("rs", colnames(dat))])

#Observed outcome levels
#Note: Replace <outcome> by metabolite, gene or protein name
y <- dat[,grep("<outcome>",colnames(dat))]

#Number of iterations
niter <- 100

#Define variables for r and r^2 for standard and robust Huber-LASSO
r_standard <- as.numeric(niter)
r2_standard <- as.numeric(niter)

r_robust <- as.numeric(niter)
r2_robust <- as.numeric(niter)


# ------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------ #
# 2. LASSO and 5-fold outcome prediction
# ------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------ #

for (i in 1:niter) {
  print(paste0("Iteration number ",i," out of ",niter))
  #Set seed
  set.seed(i)
  #Draw samples (for subset definition)
  samples <- sample(1:nrow(X), nrow(X))
  #Define subsets
  fold1 <- samples[1:(round(0.2 * nrow(X), 0))]
  fold2 <- samples[(round(0.2 * nrow(X), 0) + 1):(round(0.4 * nrow(X), 0))]
  fold3 <- samples[(round(0.4 * nrow(X), 0) + 1):(round(0.6 * nrow(X), 0))]
  fold4 <- samples[(round(0.6 * nrow(X), 0) + 1):(round(0.8 * nrow(X), 0))]
  fold5 <- samples[(round(0.8 * nrow(X), 0) + 1):nrow(X)]

  #Define variables for standard and robust prediction result
  pred_standard <- numeric(nrow(X))
  pred_robust <- numeric(nrow(X))

  #LASSO and prediction for each subset
  #Fold1
  cv_standard <-
    cv.hqreg(X[-fold1, ], y[-fold1], method = "ls", FUN = "hqreg_raw", seed = 111)
  cv_robust <-
    cv.hqreg(X[-fold1, ], y[-fold1], method = "huber", FUN = "hqreg_raw", seed = 111)

  #Prediction of Fold1
  pred_standard[1:(round(0.2 * nrow(X), 0))] <-
    predict(cv_standard, X[fold1, ], lambda = "lambda.min")
  pred_robust[1:(round(0.2 * nrow(X), 0))] <-
    predict(cv_robust, X[fold1, ], lambda = "lambda.min")

  #Fold2
  cv_standard <-
    cv.hqreg(X[-fold2, ], y[-fold2], method = "ls", FUN = "hqreg_raw", seed = 111)
  cv_robust <-
    cv.hqreg(X[-fold2, ], y[-fold2], method = "huber", FUN = "hqreg_raw", seed = 111)
```

```
#Prediction of Fold2
  pred_standard[(round(0.2 * nrow(X), 0) + 1):(round(0.4 * nrow(X), 0))] <-
    predict(cv_standard, X[fold2, ], lambda = "lambda.min")
  pred_robust[(round(0.2 * nrow(X), 0) + 1):(round(0.4 * nrow(X), 0))] <-
    predict(cv_robust, X[fold2, ], lambda = "lambda.min")

  #Fold3
  cv_standard <-
    cv.hqreg(X[-fold3, ], y[-fold3], method = "ls", FUN = "hqreg_raw", seed = 111)
  cv_robust <-
    cv.hqreg(X[-fold3, ], y[-fold3], method = "huber", FUN = "hqreg_raw", seed = 111)

  #Prediction of Fold3
  pred_standard[(round(0.4 * nrow(X), 0) + 1):(round(0.6 * nrow(X), 0))] <-
    predict(cv_standard, X[fold3, ], lambda = "lambda.min")
  pred_robust[(round(0.4 * nrow(X), 0) + 1):(round(0.6 * nrow(X), 0))] <-
    predict(cv_robust, X[fold3, ], lambda = "lambda.min")

  #Fold4
  cv_standard <-
    cv.hqreg(X[-fold4, ], y[-fold4], method = "ls", FUN = "hqreg_raw", seed = 111)
  cv_robust <-
    cv.hqreg(X[-fold4,], y[-fold4], method = "huber", FUN = "hqreg_raw", seed = 111)

  #Prediction of Fold4
  pred_standard[(round(0.6 * nrow(X), 0) + 1):(round(0.8 * nrow(X), 0))] <-
    predict(cv_standard, X[fold4, ], lambda = "lambda.min")
  pred_robust[(round(0.6 * nrow(X), 0) + 1):(round(0.8 * nrow(X), 0))] <-
    predict(cv_robust, X[fold4, ], lambda = "lambda.min")

  #Fold5
  cv_standard <-
    cv.hqreg(X[-fold5,], y[-fold5], method = "ls", FUN = "hqreg_raw", seed = 111)
  cv_robust <-
    cv.hqreg(X[-fold5,], y[-fold5], method = "huber", FUN = "hqreg_raw", seed = 111)

  #Prediction of Fold5
  pred_standard[(round(0.8 * nrow(X), 0) + 1):nrow(X)] <-
    predict(cv_standard, X[fold5,], lambda = "lambda.min")
  pred_robust[(round(0.8 * nrow(X), 0) + 1):nrow(X)] <-
    predict(cv_robust, X[fold5,], lambda = "lambda.min")

  #Reorder outcome levels (for correlation)
  y_true <- y[samples]

  #Correlation of observed and predicted outcome levels
  r_standard[i] <- cor(y_true, pred_standard, method="kendall")
  r_robust[i] <- cor(y_true, pred_robust, method="kendall")
}

#Fisher-consistent version of correlation
transf <- function(r){sin(0.5*pi*r)}
r_standard_transf <- numeric(length(r_standard))
r_robust_transf <- numeric(length(r_robust))
for (i in 1:length(r_standard)){
    r_standard__transf[i] <- transf(r_standard[i])
    r_robust_transf[i] <- transf(r_robust[i])
  }

#Squared correlation
r2_standard_transf<- numeric(length(r_standard_transf))
r2_robust_transf<- numeric(length(r_robust_transf))
for (i in 1:length(r_standard)){
    r2_standard_transf[i] <- r_standard_transf[i]^2
    r2_robust_transf[i] <- r_robust_transf[i]^2
  }


# ------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------ #
# 3. Diagnostic plots
# ------------------------------------------------------------------------------------ #
# ------------------------------------------------------------------------------------ #

#Set working directory
setwd(dir.res)

#Create plot as .tif with resolution equal to 600dpi

#Open graphic device
tiff("Figure3.tiff", height = 8, width = 7.5, units = 'in', compression="lzw", res=1200)

#Plot
boxplot(r2_standard_transf,r2_robust_transf,
        boxcol=c("black","blue"), medcol=c("black","blue"), whiskcol=c("black","blue"),
        staplecol=c("black","blue"), outcol=c("black","blue"),
        at=c(1:2),
        names=c("Standard LASSO","Robust Huber-LASSO"),
        ylab=""
)
title(ylab="Squared correlation between observed and predicted metabolite levels",
      line=2.5)

#Close device
dev.off()
```