Bioinformatics III

Seventh Assignment

Alexander Flohr (2549738)
Andrea Kupitz (2550260)

June 12, 2018

Exercise 7.1: Missing Data Imputation

Listing 1: Listing of source code

(a)
```
# Read in data table
DFrame <- read.table(file = "ms_toy.txt", sep = "\t", header = TRUE)
DTable0 <- t(as.matrix(as.data.frame(lapply(DFrame[,1:1], as.numeric))))

# create new dataset without missing data
newdata <- na.omit(DFrame)
DTable <- t(as.matrix(as.data.frame(lapply(newdata[,1:1], as.numeric))))
colnames(DTable) <- colnames(DFrame)[1]

# Calculate the mean and standard deviation of the current data
mean <- mean(DTable)
sd <- sd(DTable)

# Derive the new mean and standard deviation for the missing data based on the current
# distribution
nansize <- dim(DTable0)[1]-dim(DTable)[1]
lowquan <- qnorm(0.25, mean, sd)
lowdata <- subset(DTable, DTable[,1] <= lowquan)
newmean <- mean(lowdata)
newsd <- sd(lowdata)

# Generate the new data based on the new mean and standard deviation from the previous step
imputdata <- rnorm(nansize, newmean, newsd)

hist1 <- hist(DTable, col="blue")
hist2 <- hist(imputdata, add=T, col="red")
#plot()
for(mean in seq(from = 20, to = lowquan, by = 2)){
  for(sd in seq(from = 1, to = sd, by = 0.2)){
    imputdata1 <- rnorm(nansize, mean, sd)
    hist3 <- hist(DTable, col="blue")
    hist4 <- hist(imputdata1, add=T, col="red")
  }
}
```

Listing 1 shows the source code of our imputation of data. Figure 1 shows the imputed data.

(b) The greater the standard deviation gets, the wider gets the histogram of the imputated data. Moreover for greater standard deviations the values tend to lay more far away from each other and thus the single bars of the histogram are lower.
The closer the new mean moves to the original mean, the more moves the distribution of the imputated data to the center of the original data.
Figures 2, 3, 4 show the imputated data with increasing mean.

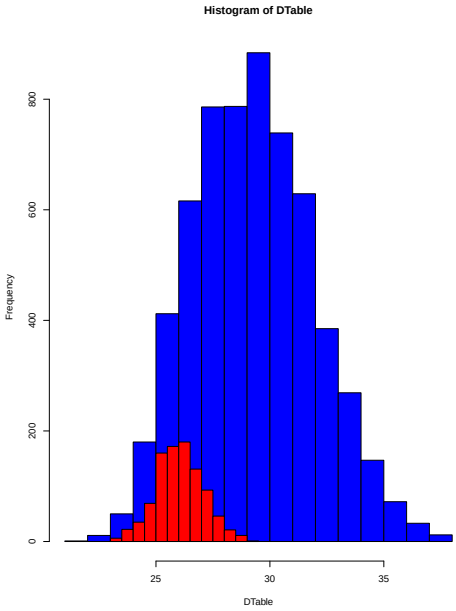

Figure 1: Distribution of the sample with the imputed data

Figures 5, 6, 7 show the imputated data with increasing standard deviation.

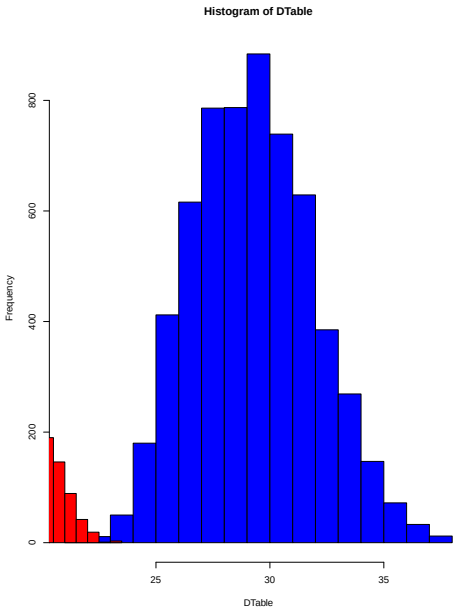

Figure 2: Distribution of the sample with the imputed data

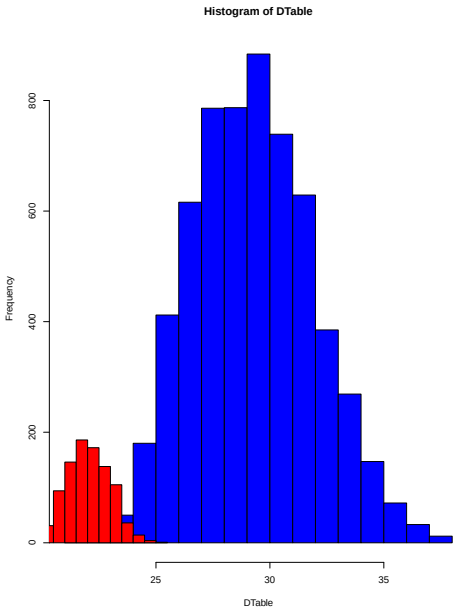

Figure 3: Distribution of the sample with the imputed data

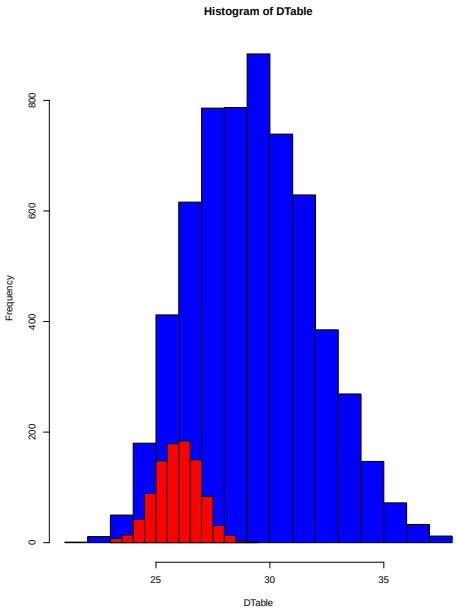

Figure 4: Distribution of the sample with the imputed data

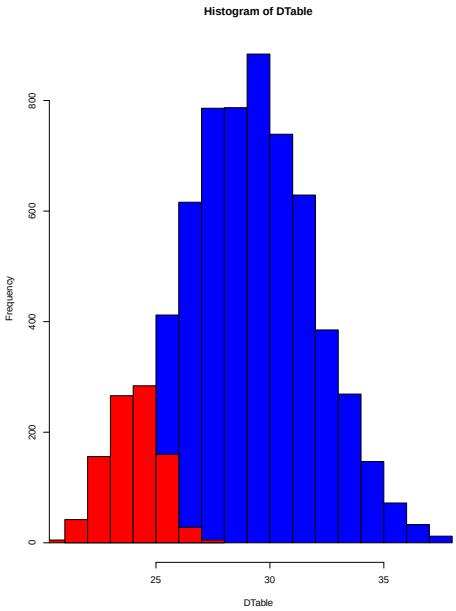

Figure 5: Distribution of the sample with the imputed data

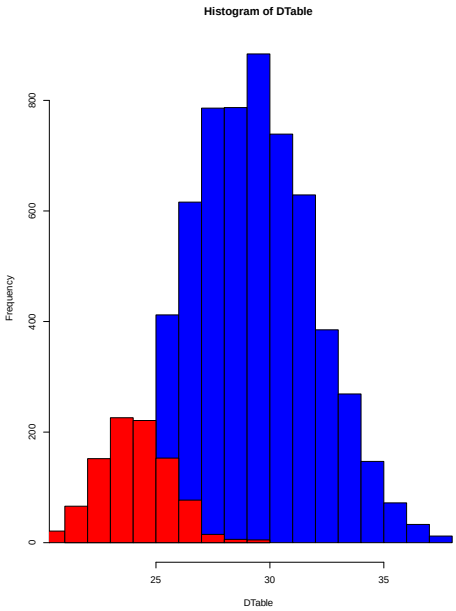

Figure 6: Distribution of the sample with the imputed data

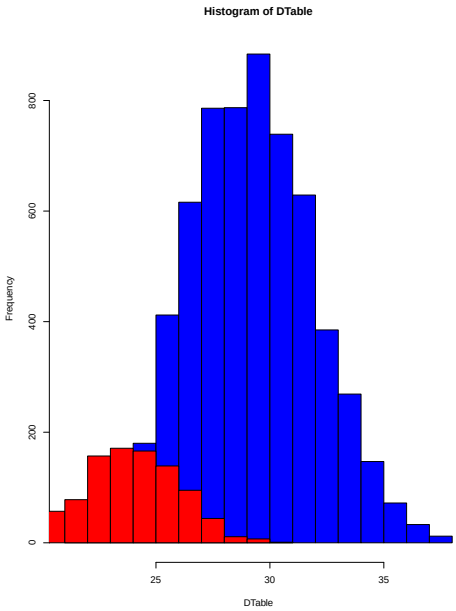

Figure 7: Distribution of the sample with the imputed data

Exercise 7.2: DREAM challenge

Listing 2 shows the R implementation of the noise modle. We used 1000 iterations to refine the modle.

Listing 2: Listing of source code

```
# Read in null-mutants.tsv as numeric matrix
df_null_mutants = read.table(
  file = "null-mutants.tsv",
  header = TRUE,
  stringsAsFactors = TRUE,
  sep = "\t"
)
rownames(df_null_mutants) = df_null_mutants$strain
mtrx = data.matrix(df_null_mutants[2:ncol(df_null_mutants)])

# global container storing the variances per genes
variance <- rep(0, ncol(mtrx))
# global container storing the probabilities
mtrx_prob <- mtrx[2:nrow(mtrx), ]

# initialize genes varances for mutants from wildtype
initializeVariance <- function() {
  for (j in 1:ncol(mtrx)) {
    temp <- 0
    for (i in 2:nrow(mtrx)) {
      temp <- temp + abs(mtrx[i, j] - mtrx[1, j])
    }
    # use variance vector as global accessible variable
    variance[j] <<- temp / (nrow(mtrx_prob))
  }
}

# update the variances for each gene based on the
# equation of noise modle step 2
# since we previousely used variances for each gene
# we interpret the equation in the same way
updateVariance <- function(p_from, p_to) {
  curr_variance <- rep(0, ncol(mtrx))
  counter <- rep(0, ncol(mtrx))
  for (i in 1:length(p_from)) {
    square_dist <- mtrx[p_from[i], p_to[i]] - mtrx[1, p_to[i]]
    square_dist <- square_dist**2
    curr_variance[p_to[i]] <- curr_variance[p_to[i]] + square_dist
    counter[p_to[i]] <- counter[p_to[i]] + 1
  }
  for(i in 1:length(curr_variance)){
    if(counter[i] != 0){
      variance[i] <<- curr_variance[i]/(1+counter[i])
    }
  }
}

# compute the probabilities that the deviation is due
# to a real regulation event
probabilities <- function() {
  for (b in 1:nrow(mtrx_prob)) {
    for (a in 1:ncol(mtrx_prob)) {
      deviation <- abs(mtrx[b + 1, a] - mtrx[1, a])
      phi <- pnorm(deviation / sqrt(variance[a]))
      mtrx_prob[b, a] <<- 2 * phi - 1
    }
  }
}

# re-estimate wilde-type expression level
# accounts to step 3 of the noise modle
```

```
updateWildType <- function(p_from, p_to){
  for(i in 1:ncol(mtrx)){
    temp <- mtrx[1,i]
    counter <- 0
    for(j in 1:length(p_from)){
      if(p_to[j] == i){
        temp <- temp + mtrx[p_from[j]+1,i]
        counter <- counter + 1
      }
    }
    mtrx[1,i] <<- temp/(counter+1)
  }
}

refine <- function() {
  # Noise model iterative step (1)
  # search gene pairs for refining the error modle
  p_from <<- c()
  p_to <<- c()
  for (b in 1:nrow(mtrx_prob)) {
    for (a in 1:ncol(mtrx_prob)) {
      if (mtrx_prob[b, a] < 0.05) {
        p_from <- c(p_from, b)
        p_to <- c(p_to, a)
      }
    }
  }
  # Noise model iterative step (2)
  updateVariance(p_from, p_to)
  # Noise model iterative step (3)
  updateWildType(p_from, p_to)
}

# sort matrices by last column
sortMtrx <- function(m){
  for(i in 2:nrow(m)){
    if(m[i,3] < m[i-1,3]){
      temp = m[i,]
      m[i,] = m[i-1,]
      m[i-1,] <- temp
      m = sortMtrx(m)
    }
  }
  return(m)
}

# output function
# looks more complicated than it is
output <- function(){
  counter <- 0
  for (b in 1:nrow(mtrx_prob)) {
    for (a in 1:ncol(mtrx_prob)) {
      if (mtrx_prob[b, a] < 0.05) {
        counter <- counter + 1
      }
    }
  }
  outcome <- matrix(NA, nrow = counter, ncol = 3)
  colnames(outcome) <- c("Gene_A", "Gene_B", "Probability")
  counter <- 1
  for (b in 1:nrow(mtrx_prob)) {
    for (a in 1:ncol(mtrx_prob)) {
      if (mtrx_prob[b, a] < 0.05) {
        outcome[counter, 1] <- b
        outcome[counter, 2] <- a
        outcome[counter, 3] <- round(mtrx_prob[b,a], digits = 8)
        counter <- counter + 1
```

```
      }
    }
  }
  outcome = sortMtrx(outcome)
  for(i in 1:nrow(outcome)){
    cat("G", outcome[i,1], "\tG", outcome[i,2],
        "\t", outcome[i,3], "\n", sep = "")
  }
}

# Main
iterations <- 1000
initializeVariance()

for (i in 1:iterations) {
  # Steps of Noise model
  # calculate probabilities for each pair of genes
  probabilities()
  # refine modle
  refine()
}
output()
```

Output:

```
G7 G10 0
G4 G6 0.00282553
G2 G6 0.01111409
G3 G6 0.01395305
G6 G5 0.02255766
G10 G5 0.02255766
G5 G6 0.02788807
G6 G7 0.04557906
G8 G7 0.04557906
```

Less iterations could increase the number of predicted links bout might be less precise.