Batch effect correction with sample remeasurement in case-control studies

Overview

The `BatchReMeasure` package provides methods for correcting batch effects by remeasuring control samples. Currently, we consider the control and case samples collected in the first and second batches, respectively, and a subset of control samples are remeasured in the second batch. We maximize the joint model to obtain estimates of parameters like the true and batch effects. In addition, this package can evaluate the power of the developed procedure and provide a power calculation tool.

```
# rmarkdown::html_vignette
library(BatchReMeasure)
```

Example with simulated data

The following script estimates the true and batch effects on simulated data. We have both the R-version remeasure functions and the more computationally efficient C++ wrapper. For example, we set sample size $n = 100$ and remeasured sample size $n_1 = 40$. Samples consist of 50 control samples and 50 case samples. Set the std of the first batch as 2 and the std of the second batch as 1; The correlation, true effect, and batch effect are set as $0.6, 0.5$, and 0.5, respectively.

```
n = 100
n1 = 40
r1 = 2  # Variance for batch 2;
# variance for batch 1 is set to 1, then the true
# effect corresponds to the cohen's d.
r2 = 0.6  # Inter-batch correlation
a0 = 0.5  # true effects
a1 = 0.5  # batch effects
```

We then simulate the data, where `Y` is the response vector for all control and case samples, `Z` is the design matrix, and `ind.r` is the index for control samples remeasured in the first batch. `Y.r` is the response vector for remeasured samples whose length is less than or equal to the control sample size.

```
X = as.numeric(gl(2, n/2)) - 1
Z <- cbind(rep(1, n), rnorm(n))
b <- c(0, -0.5)
v1 = r1^2
v2 = 1
Et <- rnorm(n, sd = ifelse(X == 0, sqrt(v1), sqrt(v2)))
Y <- Z %*% b + cbind(X, X) %*% c(a0, a1) + Et
Z.r.a <- Z[1:(n/2), ]
Et.r.a <- Et[1:(n/2)]
Y.r.a <- a1 + Z.r.a %*% b + r2 * sqrt(v2) * Et.r.a/sqrt(v1) +
    rnorm(n/2, sd = sqrt((1 - r2^2) * v2))
ind.r <- 1:n1
Y.r = Y.r.a[ind.r]
```

Since the batch effects and the biological effects are indistinguishable in this example, we conduct batch effect correction with remeasurement using the `batch.ReMeasure.S1` function. We obtain the estimated true and batch effects, and the p-value for the correction.

```
Estimate = batch.ReMeasure.S1(Y, X, Z, ind.r, Y.r)
a0H = Estimate$a0
a0Var = Estimate$a0Var
a1H = Estimate$a1
pv_re = Estimate$p.value
pv_re
#> [1] 0.01319093
```

The package also includes functions that only consider case and remeasured control samples in the second batch and ignore batch effects.

```
Estimate = batch.Batch2.S1(Y, X, Z, ind.r, Y.r)
pv_b2 = Estimate$p.value
pv_b2
#>            [,1]
#> [1,] 0.02032435
```

If we ignore the batch effect, simply run

```
Estimate = batch.Ignore.S1(Y, X, Z)
pv_ignore = Estimate$p.value
pv_ignore
#>              [,1]
#> [1,] 2.341729e-07
```

Additionally, we can use the residual bootstrap to improve the estimation of uncertainty.

```
# The residual bootstrap
boot = batch.ReMeasure.S1.res(Y, X, Z, ind.r, Y.r)
#> [1] 200
#> [1] 400
#> [1] 600
#> [1] 800
#> [1] 1000
ztest = boot$ztest
ztest_b = boot$ztest_b
```

Real data example: ovarian cancer dataset

We will use the ovarian cancer dataset, which is available in our package. In this dataset, we have four cancer subtypes: C1-MES, C2-IMM, C4-DIF, and C5-PRO. The gene expression was profiled in two different platform: Agilent and RNASeq. This dataset includes gene expression data for four cancer subtypes: C1-MES, C2-IMM, C4-DIF, and C5-PRO. Gene expression was profiled using two different platforms: Agilent and RNASeq. We will demonstrate how to preprocess the data and correct batch effects using the `batch.ReMeasure.S1` function.

```
data("Agilent.dat")
data("Agilent.recur")
data("Agilent.subtype")
data("RNAseq.dat")
data("RNAseq.recur")
data("RNAseq.subtype")
```

```r
# C1-MES
Agilent_C1MES_name = colnames(Agilent.dat[, which((Agilent.subtype == "C1-MES") %in%
    TRUE)])
RNAseq_C1MES_name = colnames(RNAseq.dat[, which((RNAseq.subtype == "C1-MES") %in%
    TRUE)])
ReMeasure_C1MES_name = rownames(remeasure.subtype[which((remeasure.subtype ==
    "C1-MES") %in% TRUE), , drop = F])
C1MES_Agilent = Agilent.dat[, Agilent_C1MES_name]
C1MES_RNAseq = RNAseq.dat[, RNAseq_C1MES_name]
dim(C1MES_Agilent)
#> [1] 11861    76
dim(C1MES_RNAseq)
#> [1] 11861    25
# C2-IMM
Agilent_C2IMM_name = colnames(Agilent.dat[, which((Agilent.subtype == "C2-IMM") %in%
    TRUE)])
RNAseq_C2IMM_name = colnames(RNAseq.dat[, which((RNAseq.subtype == "C2-IMM") %in%
    TRUE)])
ReMeasure_C2IMM_name = rownames(remeasure.subtype[which((remeasure.subtype ==
    "C2-IMM") %in% TRUE), , drop = F])
C2IMM_Agilent = Agilent.dat[, Agilent_C2IMM_name]
C2IMM_RNAseq = RNAseq.dat[, RNAseq_C2IMM_name]
dim(C2IMM_Agilent)
#> [1] 11861    77
dim(C2IMM_RNAseq)
#> [1] 11861    20
# C4-DIF
Agilent_C4DIF_name = colnames(Agilent.dat[, which((Agilent.subtype == "C4-DIF") %in%
    TRUE)])
RNAseq_C4DIF_name = colnames(RNAseq.dat[, which((RNAseq.subtype == "C4-DIF") %in%
    TRUE)])
ReMeasure_C4DIF_name = rownames(remeasure.subtype[which((remeasure.subtype ==
    "C4-DIF") %in% TRUE), , drop = F])
C4DIF_Agilent = Agilent.dat[, Agilent_C4DIF_name]
C4DIF_RNAseq = RNAseq.dat[, RNAseq_C4DIF_name]
# C5-PRO
Agilent_C5PRO_name = colnames(Agilent.dat[, which((Agilent.subtype == "C5-PRO") %in%
    TRUE)])
RNAseq_C5PRO_name = colnames(RNAseq.dat[, which((RNAseq.subtype == "C5-PRO") %in%
    TRUE)])
ReMeasure_C5PRO_name = rownames(remeasure.subtype[which((remeasure.subtype ==
    "C5-PRO") %in% TRUE), , drop = F])
C5PRO_Agilent = Agilent.dat[, Agilent_C5PRO_name]
C5PRO_RNAseq = RNAseq.dat[, RNAseq_C5PRO_name]
```

The Agilent platform has more samples (around 75 for each subtype) than RNASeq (about 20 for each subtype). For future use, we will bind three of the four subtypes into groups (e.g., C2+C4+C5, C1+C2+C5, C1+C2+C4, C1+C4+C5). To mimic a study design, we will compare one subtype against the other three. Additionally, for each combination, a common set of remeasured samples are available. For example, we have 36 samples that are profiled in both the Agilent and RNASeq platforms, as well as similar sets for other combinations.

```r
C2C4C5_Agilent = cbind(C2IMM_Agilent, C4DIF_Agilent, C5PRO_Agilent)
C2C4C5_RNAseq = cbind(C2IMM_RNAseq, C4DIF_RNAseq, C5PRO_RNAseq)
```

```
ReMeasure_C2C4C5_name = c(ReMeasure_C2IMM_name, ReMeasure_C4DIF_name,
    ReMeasure_C5PRO_name)
length(ReMeasure_C2C4C5_name)
#> [1] 36

C1C2C5_Agilent = cbind(C1MES_Agilent, C2IMM_Agilent, C5PRO_Agilent)
C1C2C5_RNAseq = cbind(C1MES_RNAseq, C2IMM_RNAseq, C5PRO_RNAseq)
ReMeasure_C1C2C5_name = c(ReMeasure_C1MES_name, ReMeasure_C2IMM_name,
    ReMeasure_C5PRO_name)
length(ReMeasure_C1C2C5_name)
#> [1] 35

C1C2C4_Agilent = cbind(C1MES_Agilent, C2IMM_Agilent, C4DIF_Agilent)
C1C2C4_RNAseq = cbind(C1MES_RNAseq, C2IMM_RNAseq, C4DIF_RNAseq)
ReMeasure_C1C2C4_name = c(ReMeasure_C1MES_name, ReMeasure_C2IMM_name,
    ReMeasure_C4DIF_name)
length(ReMeasure_C1C2C4_name)
#> [1] 36

C1C4C5_Agilent = cbind(C1MES_Agilent, C4DIF_Agilent, C5PRO_Agilent)
C1C4C5_RNAseq = cbind(C1MES_RNAseq, C4DIF_RNAseq, C5PRO_RNAseq)
ReMeasure_C1C4C5_name = c(ReMeasure_C1MES_name, ReMeasure_C4DIF_name,
    ReMeasure_C5PRO_name)
length(ReMeasure_C1C4C5_name)
#> [1] 34
# Agilent_Re = Agilent.dat[ , rownames(remeasure.recur)]
# RNAseq_Re = RNAseq.dat[ , rownames(remeasure.recur)]
```

We then compare our approach, `batch.ReMeasure.S1`, to two most influential methods like `SVA` and `ComBat`. We will obtain p-values for the correction and adjust them using the Benjamini-Hochberg procedure. Here we only show C1+C2+C5 RNAseq vs. C4 Agilent. We have remeasured samples in C1+C2+C5 Agilent. If users want to see the results of other combinations, just change the `Datalist` in the code chunk below

Our findings indicate that `ComBat` is too conservative, while `sva` underadjusts the batch effects. In contrast, our method, `batch.ReMeasure.S1`, provides an accurate and efficient way to correct batch effects in case-control studies using sample remeasurement.

```
# library(ggplot2) library(cowplot) library(reshape) allMethods =
# c('ReMeasure', 'ComBat', 'sva') n1s = seq(10, length(Ind_all), by =
# 5) nGene = 11861 procedure = 'BH' res.pv.a = array(NA,
# c(length(allMethods), length(n1s), nGene), dimnames = list(Method =
# allMethods, ReMeasureNo = paste(n1s), Gene = paste(1:nGene) ) )
# create.data = function(res.pv.a, procedure) { res.df <-
# reshape::melt(res.pv.a) colnames(res.df)[ncol(res.df)] <- 'Value'
# if (procedure == 'none') { m <- aggregate(Value~Method +
# ReMeasureNo, res.df, function(x) sum(x <= alpha )) } else if
# (procedure == 'BH') { m <- aggregate(Value~Method + ReMeasureNo,
# res.df, function(x) sum( p.adjust(x, method = 'BH') < alpha ) ) }
# else if (procedure == 'bonferroni') { m <- aggregate(Value~Method +
# ReMeasureNo, res.df, function(x) sum( p.adjust(x, method =
# 'bonferroni') < alpha ) ) } m$ReMeasureNo = factor(m$ReMeasureNo)
# return(m) } for (m in 1:length(allMethods)) { method =
# allMethods[m] if (method == 'ReMeasure') { pvMat = res$pv } else if
# (method == 'ComBat') { pvMat = res$pv_par } else if (method ==
# 'sva') { pvMat = res$pv_sva } for (num in 1:length(n1s)) { n1 =
# n1s[num] res.pv.a[method, as.character(n1), ] = pvMat[num, ] } }
# alpha = 0.05 res.df = create.data(res.pv.a, procedure = procedure)
# res.df$Method <- factor(res.df$Method, levels = c('ReMeasure',
# 'ComBat', 'sva', 'LMM')) str = stri_replace_all_regex(Datalist,
# pattern =c('MES', 'IMM', 'PRO', 'DIF', '_'), replacement =
# c('','','','', ' '), vectorize = FALSE) str =
# sub('(C\\d)(C\\d)(C\\d)', '\\1+\\2+\\3+', str) obj <-
# ggplot(res.df, aes(x = ReMeasureNo, y = Value+1, group = Method,
# col = Method, shape = Method, linetype = Method)) +
# geom_line(linewidth = 1) + geom_point(size = 2) obj <- obj +
# xlab('No. of remeasured samples') + theme_bw(base_size = 16) +
# theme(legend.position='bottom', axis.text.x = element_text(angle =
# 45, hjust=1)) + ggtitle(gsub('_', ' ', paste(str[-3], collapse = '
# vs. ') ) ) + theme(plot.title = element_text(size = 15, hjust =
# 0.5)) + ylab('No. of discoveries') + theme(legend.title =
# element_blank(), panel.grid.minor = element_blank()) print(obj)
```

Conclusion

The BatchReMeasure package provides a convenient tool for correcting batch effects in case-control studies using sample remeasurement. By doing so, it enables accurate downstream analysis of the data. We hope that this example demonstrates the utility of our package in practical applications.