Remove Batch Effect Report

Ryan Gallagher

2025-03-31

Purpose

In Data Analysis meetings, we have been discussing ideas surrounding the application of batch effect correction to our large Knock-down Experiments (EditCo) dataset.

This report will outline the method used in the function `limma::removeBatchEffect()` and discuss it's implications in our data. This report will also discuss how the authors of `DESeq2` recommend designing around batches. Finally, we will give the function a try at our data and see exactly what the output is.

limma::removeBatchEffect()

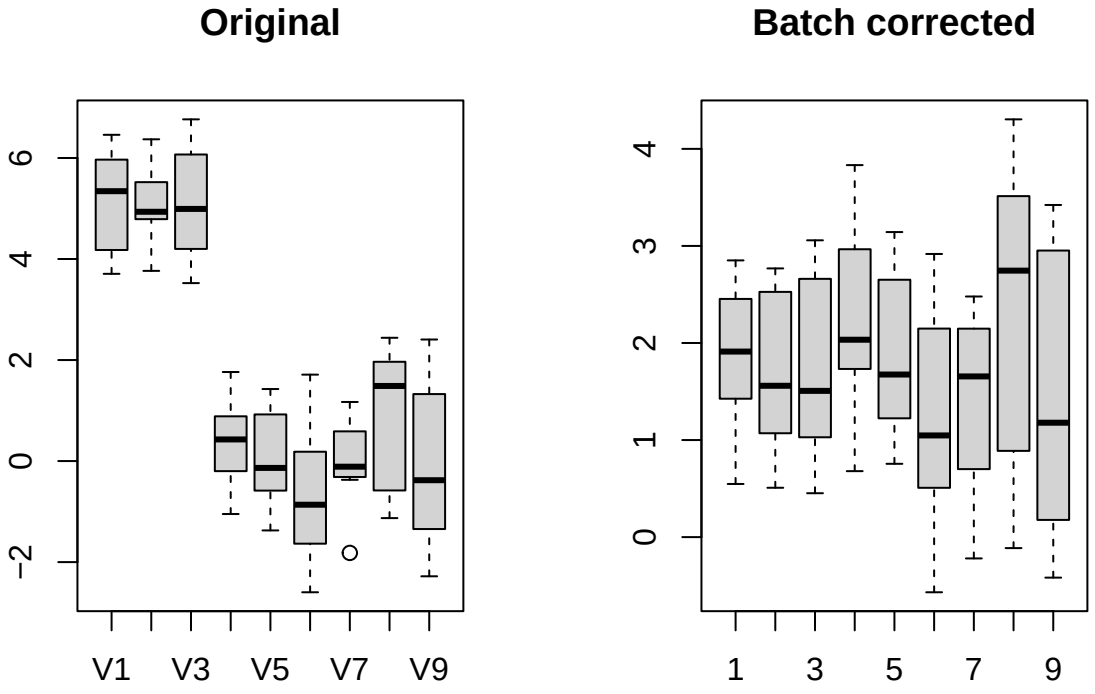
Intro

The documentation details:

> *This function is useful for removing unwanted batch effects, associated with hybridization time or other technical variables, ready for plotting or unsupervised analyses such as PCA, MDS or heatmaps. The design matrix is used to describe comparisons between the samples, for example treatment effects, that should not be removed. The function (in effect) fits a linear model to the data, including both batches and regular treatments, then removes the component due to the batch effects.*
>
> *In most applications, only the first batch argument will be needed. This case covers the situation where the data has been collected in a series of separate batches.*

With an example:

```
y <- matrix(rnorm(10*9),10,9)
y[,1:3] <- y[,1:3] + 5
batch <- c("A","A","A","B","B","B","C","C","C")
y2 <- removeBatchEffect(y, batch)
par(mfrow=c(1,2))
boxplot(as.data.frame(y),main="Original")
boxplot(as.data.frame(y2),main="Batch corrected")
```



Under-the-Hood

The function `limma::removeBatchEffect()` employs linear modeling to identify and subtract batch-associated variation from our data. This function accepts an expression matrix (E), batch information (Z) and an optional design matrix to model specific experimental conditions or treatment effects (X) that should be preserved during batch effect removal.

This process works as follows:

1. Format your expression matrix and define your batch information / design matrix.
2. Per gene, fit two linear models for your expression data
   - **Full Model:** $Y_f = X\beta + Z\alpha$
   - **Reduced Model:** $Y_r = X\beta$

   1. Where:
      - X = Design Matrix for Experimental Condition(s) to be retained
      - β = Coefficients for Experimental Condition(s)
      - Z = Design Matrix for Batch Effect(s)
      - α = Coefficients for Batch Effect(s)
3. Obtain your Full & Reduced model matrices of predicted outcomes per gene.
4. At each gene, subtract the Reduced model predicted results from the Full model predicted results to **isolate** the component that is due to batch effect to obtain a new Predicted Batch Effect matrix.
   - $Y_f - Y_r = X\beta + Z\alpha - X\beta = Z\alpha = Y_Z$
   - Y_Z = Predicted Batch Effect Matrix
5. Subtract the Batch Effect from the original expression matrix (E)
   1. Adjusted Expression = $E - Y_Z$
   2. **This is the final output from this linear model fitting method.**

Pitfalls in our Data

The `DESeq2` vignette explicitely outlines a common issue individuals face when trying to adjust for batch effect within their design, and this issue can be extended towards a manual removal of batch effect via other methods:

"Model Matrix not full rank"

> "While most experimental designs run easily using design formula, some design formulas can cause problems and result in the *DESeq* function returning an error with the text: "the model matrix is not full rank, so the model cannot be fit as specified." There are two main reasons for this problem: either one or more columns in the model matrix are linear combinations of other columns, or there are levels of factors or combinations of levels of multiple factors which are missing samples. We address these two problems below and discuss possible solutions:"

Linear Combinations

> "The simplest case is the linear combination, or linear dependency problem, when two variables contain exactly the same information, such as in the following sample table. The software cannot fit an effect for `batch` and `condition`, because they produce identical columns in the model matrix. This is also referred to as *perfect confounding*. A unique solution of coefficient is not possible."

```
## DataFrame with 4 rows and 2 columns
##      batch condition
##   <factor>  <factor>
## 1        1         A
## 2        1         A
## 3        2         B
## 4        2         B
```

> "Another situation which will cause problems is when the variables are not identical, but one variable can be formed by the combination of other factor levels. In the following example, the effect of batch 2 vs 1 cannot be fit because it is identical to a column in the model matrix which represents the condition C vs A effect."

```
## DataFrame with 6 rows and 2 columns
##      batch condition
##   <factor>  <factor>
## 1        1         A
## 2        1         A
## 3        1         B
## 4        1         B
## 5        2         C
## 6        2         C
```

> "In both of these cases above, the batch effect cannot be fit and must be removed from the model formula. There is just no way to tell apart the condition effects and the batch effects. The options are either to assume there is no batch effect (which we know is highly unlikely given the literature on batch effects in sequencing datasets) or to repeat the experiment and properly balance the conditions across batches. A balanced design would look like:"

```
## DataFrame with 6 rows and 2 columns
##      batch condition
##   <factor>  <factor>
## 1        1         A
## 2        1         B
## 3        1         C
```

```
## 4       2         A
## 5       2         B
## 6       2         C
```

An Attempt Anyways

The following code outlines an attempt I made at using `limma::removeBatchEffect`. We define a `Plate` variable that identifies which plate the experiment was run on. We also define a `gene` variable (our `treatment`) that defines our experiments and their replicates.

We will find that we cannot estimate our batching coefficients and our resultant matrix will be identical to the input.

```
library(tidyverse)
library(DESeq2)
# Read in combined counts data w. meta
cts = as.matrix(read.csv("./data/counts.csv", row.names = 1))
meta = read.csv("./data/meta.csv") # Info gathered from sample name

#Check that the order is the same (requirement of DESeq2)
all(meta$sample == colnames(cts))
```

```
## [1] TRUE
```

```
## Normalize
### Make meta to distinguish WT / TRAC across plates
meta.norm = meta %>%
  mutate(gene = ifelse(condition == "control", paste(gene, plate, sep = "_"), gene))
cts.norm = cts

# Make DESeq2 object
dds = DESeqDataSetFromMatrix(countData = cts.norm, colData = meta.norm,
                             design = ~ gene)
```

```
## Warning in DESeqDataSet(se, design = design, ignoreRank): some variables in
## design formula are characters, converting to factors
```

```
# remove low abundance
keep = rowSums(counts(dds) >= 10) >= 3
dds = dds[keep,]

# Try batch removal
vsd.batch = vst(dds, blind=F)
#blind=T would treat all samples as independent (we have replicates)
vsd.batch$plate = as.factor(vsd.batch$plate)
mat = assay(vsd.batch)
mm = model.matrix(~gene, colData(vsd.batch))
mat = limma::removeBatchEffect(mat, batch = vsd.batch$plate, design=mm)
```

```
## Coefficients not estimable: batch1 batch2 batch3 batch4 batch5
```

```
## Warning: Partial NA coefficients for 14186 probe(s)
```

```
lim.result = assay(vsd.batch)

# If TRUE, then the limma adjusted data is that same as the input data.
all(lim.result == assay(vsd.batch))
```

```
## [1] TRUE
```

Some References

Responses from the Authors on Help Forums for this Issue:

1. https://www.biostars.org/p/266507/
2. https://support.bioconductor.org/p/111119/
3. https://www.reddit.com/r/bioinformatics/comments/1cj6ymm/removebatcheffect_explained_using_base_r_linear/
4. https://support.bioconductor.org/p/133203/