Sample DGE Workflows

John Thompson

December 28, 2015

Contents

1 Overview of DGE Workflow **3**

1.1 Normalization: runEdgeRNorm . . . 3

1.2 Model Fit: runVoom . . . 3

1.3 Detrending Artifacts: runSVA . . . 3

1.4 SLOA Composition after runVoom: . . . 4

1.5 Contrasts: runContrasts . . . 4

1.6 Workflow Diagram . . . 5

2 Test Dataset : IPF Fibroblasts from Stable and Rapid progressing donors. **6**

3 Establish the Model **6**

3.1 Concatenated Disease/Treatment Model (RefGene) . . . 6

4 Present/Absent filtering with zFPKM **8**

5 EdgeR Dispersion Plot **9**

6 DGE Analysis **10**

6.1 EdgeR TMM Normalization . . . 10

6.2 Simple Voom . . . 11

6.3 Voom with Quality Weights . . . 11

6.4 Voom with Quality Weights and var.design . . . 12

6.5 Voom with duplicateCorrelation . . . 13

6.6 Voom Summary . . . 14

7 Run Contrasts **15**

7.1 Signature Genes Tables . . . 16

7.2 PValue Histograms . . . 18

7.2.1 Before using duplicateCorrelation . . . 18

7.2.2 After using duplicateCorrelation . . . 19

8 MDS : Multi-Dimensional Scaling **20**

9 SVA Analysis : Removing unknown confounding factors **23**

10 Evaluating TMM Normalization **26**

11 Session Info **29**

1 Overview of DGE Workflow

The DGE workflow supported by DGE.Tools in simpliest form includes these Steps:

1.1 Normalization: runEdgeRNorm

Performs edgeR TMM Normalization (or any normalization supported by calcNormFactors). Counts are converted to a DGEList object by the edgeR::DGEList, then edgeR::calcNormFactors normalizes the data. Results are returned as a SummarizedListOfArrays object (SLOA).

The SLOA object at this stage contains the following objects:

- rowData: gene annotation
- colData: Sample annotation (experiment design data)
- Counts: Unmodified gene counts
- DesignMatrix: Constructed from the supplied formula
- DGEList: contains counts and normfactors
- Formula: chr string saved exactly as passed in the argument
- QC.Metrics.Summary: Carried over to the SLOA for convenience, only if this optional item was present in the RSE object.

1.2 Model Fit: runVoom

Takes the SLOA from runEdgeRNorm and runs **VoomWithQualityWeights** by default, then runs **lmFit** using the **Formula** and **DesignMatrix** already present in the SLOA object.

There are a several options to runVoom to handle different experiment designs.

A **qualityWeights** argument allows you disable qualityWeights, although this is not recommended (except when group N is < 3). By default, quality weights evaluates each sample individually. If weights are associated with a sample factor (e.g. normal samples seem to have a different weight than tumor samples) you can place a design matrix called **var.design** into the SLOA and **voomWithQualityWeights** will use this to treat groups of samples together for quality assessment and weighting. For more information about quality weights, see Liu et.al. (NAR 5/2015). Quoting the abstract: "... this strategy leads to a universally more powerful analysis and fewer false discoveries when compared to conventional approaches."

A **dupCor** argument allows you to invoke the **duplicateCorrelation** function to deal with nested designs (see section 6.5 below). For duplicateCorrelation to work, a **block** column must be present in the **colData** object (sample annotation) of the SLOA. Operationally **voom/lmFit** runs first, then **duplicateCorrelation** is invoked on the fit, then **voom/lmFit** runs again, this time with the blocking information and correlation value provided by **duplicateCorrelation**.
duplicateCorrelation Reference

1.3 Detrending Artifacts: runSVA

SVA detrends unknown confounding factors in your data. Operationally, SVA is executed after runVoom. SVA adds additional variables it finds to the DesignMatrix. Then runVoom is executed again this time including the surrogate variables.

1.4 SLOA Composition after runVoom:

After completing runVoom, the following data objects are added to the SLOA object.

- Log2CPM: The normalized Log2CPM used by Voom/lmFit
- Fit: The fit object produces by lmFit
- Residualts: From the fit
- Elist: The voom Elist object

1.5 Contrasts: runContrasts

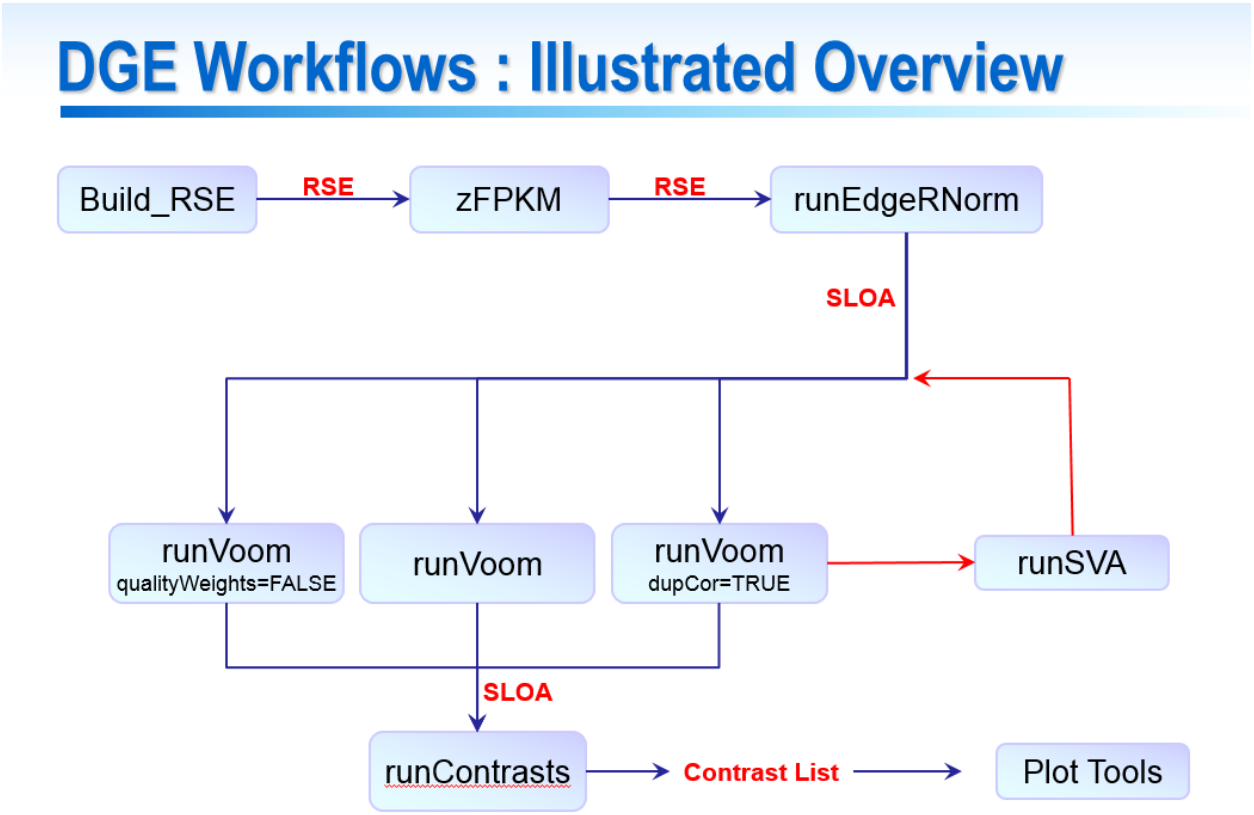
Takes the SLOA from runVoom and preforms the contrasts you define in the input contrast List using contrasts.fit. Finally, eBayes with robust=TRUE is run on the resulting fit.

The output of **runContrast** is a List object containing the following data elements:

- ContrastMatrix: From makeContrasts
- Fit.Contrasts: From contrasts.fit
- colData: Sample (Design) annotation
- rowData: Gene annotation
- Fit.Contrasts.treat: Present if eBayes was run
- TopTableList: A list of topTable data.frames for each contrast
- TopTreatList: A list of topTreat data.frames for each contrast (present if TopTreat was enabled)
- SigCountsSummary: A data.frame summary table of significant gene counts
- Created: a list of information about how the analysis was conducted
- Level: Defines the data level (one of Gene, Transcript or Exon)

1.6 Workflow Diagram

The following figure illustrates how the pieces described above fit together.



DGE_WorkFlowDiagram.png

2 Test Dataset : IPF Fibroblasts from Stable and Rapid progressing donors.

The test data selected for the tutorial is human lung fibroblast cell lines from 9 donors cultured in vitro either with or without TFGb treatment. The donors fall into three groups by disease status (Rapid, Stable or Normal). So there are essentially three factors (donor, disease status and treatment) with treatment nested within donor. We're only interested in disease and treatment effects and wish to "dial out" the effect of donor.

3 Establish the Model

This is the most important part of the whole process.

The factors of interest are Disease status and Treatment factors. To reparameterize this data in a way that simplifies the contrasts we want to perform, we'll concatenate those two factors to produce a single combined Disease-treatment factor with 6 levels.

3.1 Concatenated Disease/Treatment Model (RefGene)

Model for consideration:

- ~ 0 + DiseaseTreatment

Disease (3 levels; Norm, Rapid and Stable) and Treatment (2 levels; Veh and TGFb) will be concatenated to create one factor column with 6 levels that allows for more intuitive contrasts to be formed.

DiseaseTreatment Levels:

- TGFb_Norm
- TGFb_Rapid
- TGFb_Stable
- Veh_Norm
- Veh_Rapid
- Veh_Stable

Contrasts of Interest:

- Rapid vs Norm (no treatment)
- Stable vs Norm (no treatment)
- Rapid vs Stable ((no treatment)
- TGFb_Norm vs Veh_Norm
- TGFb_Rapid vs Veh_Rapid
- TGFb_Stable vs Veh_Stable
- TGFb_Rapid vs TGFb_Norm
- TGFb_Stable vs TGFb_Norm
- TGFb_Rapid vs TGFb_Stable

CodeBlock: Clear the workspace and load the packages we need

```
rm(list=ls()) #Clear the workspace
invisible(gc()) #garbage collection to masimize available memory
startTime = Sys.time()

library(magrittr)
library(dplyr)
library(ggplot2)
library(SummarizedExperiment)
library(DGE.Tools)
library(stringi)
library(stringr)
library(reshape2)

source("~/R/lib/SubsettableListOfArrays.R")
```

Code Block: Settings Section

In this analysis, the RSE for this dataset has already been created and is the starting point for this analysis.

```
########## Settings Section
#
# Put this RMD file in the same folder as MyGene_RSE.RDS and make that folder
# the working directory before running the markdown file.

setwd("~/Fibrosis/IPF Cedar Sinai/RNA-Seq/RefGene2015/RData")

RSE = readRDS("MyGene_RSE.RDS")

# In this example, I need to concatenate two columns to make the column I
# want to use in the formula.  If the columns you need for your formula are
# already in the Design table, you're all set and can proceed to define
# your formula. Otherwise, customize the next code block to create any columns
# you need for the formula. Any categorical columns used in a model have to
# be converted to factors in the Design table

#Customize the design table to concatenate Disease and Treatment factors
Design = colData(RSE) %>% as.data.frame

#Concatenate the two columns and convert to a factor
Design$DT = paste(Design$Disease, Design$Treatment, sep="") %>% as.factor

#put the updated design table back into the RSE
colData(RSE) = Design %>% DataFrame

#set up the formula
MyFormula = "~ 0 + DT"
ModelName = "DT_Model"  #used to name the RDS file

########## End of Settings Section
```

4 Present/Absent filtering with zFPKM

It's important to remove non-expressed genes before running the Voom pipeline. Genes not expressed are not interesting and just increase the multiple testing burden. They also foul up the pvalue distributions used to evaluate the quality of your model.

Set up several different criteria for genes above a threshold

- present.all: GeneIDs present in all samples
- present.50: GeneIDs present in 1/2 of samples
- present.any: GeneIDs present in at least one sample

You can use these later to filter dataframes or RSE objects e.g.:

mydataframe = mydataframe[present.any,] MyRSE = MyRSE[present.any,]

For this analysis we'll use the Present.50 criteria

Code Block: Removal of undetected genes

```
#Present/Absent filtering
#Use zFPKM to define some gene present thresholds
zFPKM <- assay(RSE, "zFPKM")
present <- zFPKM > -3  #boolean matrix true for each gene above threshold

#count genes present for each sample  (column sum)
samplePresent <- colSums(present) #count genes present for each sample

#boolean flag gene present in all samples
present.all <- rowSums(present) == ncol(present)
#boolean flag for genes present in 1/2 of samples
present.50 <- rowSums(present) >= (ncol(present)/2)
#boolean flag for genes present in at least one sample
present.any <- rowSums(present) >= 1

#how many genes present in any sample:
count.present.any <- sum(present.any)

#Subset RSE to contain only present genes
RSE <- RSE[present.50,]
```

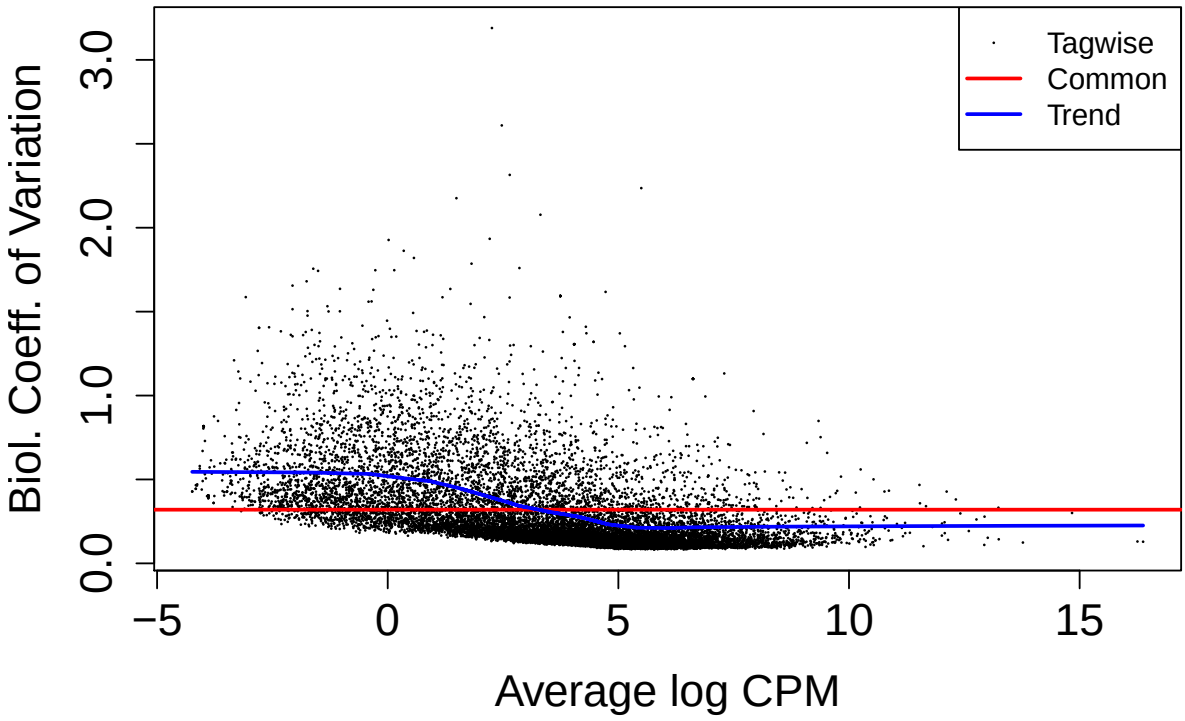
Present Gene Count = 14467

5 EdgeR Dispersion Plot

The Dispersion QC plot is generated to evaluate the overall variation in your experiment across the full intensity range.

Code Block: EdgeR Dispersion Plot

```
#This plot is the slowest piece of the whole pipeline,  I often comment the next
#line out while I'm tweaking other blocks in the workflow.
plotDispersion (assay(RSE, "Counts"), MyFormula, Design)
```




```
## [1] 0
```

The blue line is a fit line for your cloud of data. The Red line is a Common treshold. The bulk density of your data should be at or below this line for a good quality experiment. But the result is somewhat dependent on the inherent variability in your system. For example, expect lower variance from stable cell lines or inbred mice than you would from outbred humans. There's always greater variance at the low intensity end. And regulated genes show up as dots above the dense cloud across the intensity range.

6 DGE Analysis

The functions edgeRNorm, runSVA, runVoom can be used with a different options to accommodate a variety of experiment designs and artifacts that need to be dealt with.

Basically, raw counts are normalized by runEdgeRNorm first. Optionally, runSVA can be used to account for unknown factors or batch effects. Then runVoom is used to fit a model. This step can include assigning QualityWeights to each sample, and can also account nested designs using the duplicateCorrelation method.

In this tutorial, we'll run a series of analysis of increasing complexity.

- Simple voom: no quality weights, duplicate correlation or SVA analysis
- voomWithQualityWeights: Just add quality weights to the analysis
- duplicateCorrelation: Add duplicateCorrelation analysis (still useing quality weights).
- SVA analysis: Include SVA analysis in the mix

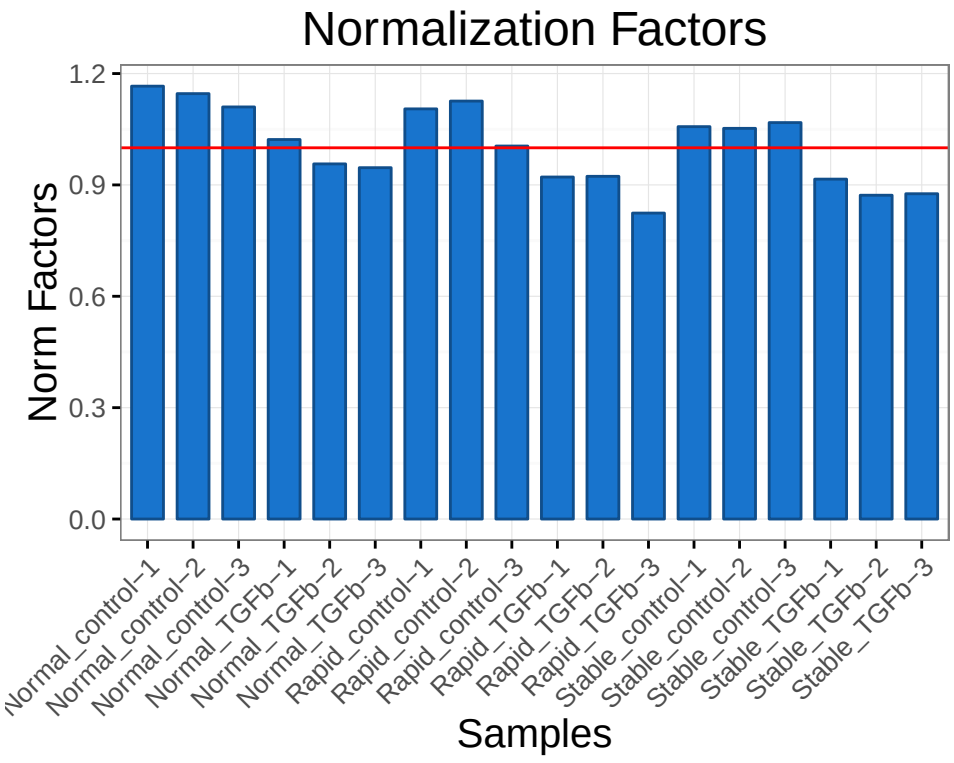
6.1 EdgeR TMM Normalization

EdgeR normaliztion is a common step to all workflow examples. We'll run edgeR with the default TMM normalization.

runEdgeRNorm requires an RSE object and a formula (from the settings section) and produces a SLOA object.

Code Block: edgeR Normalization

```
Labels = paste(colData(RSE)$ReplicateGroup, c(1:3), sep="-")
MySLOA = runEdgeRNorm(RSE, MyFormula, plotFile = "Norm.Factors.png",
                      plotLabels = Labels)
```



6.2 Simple Voom

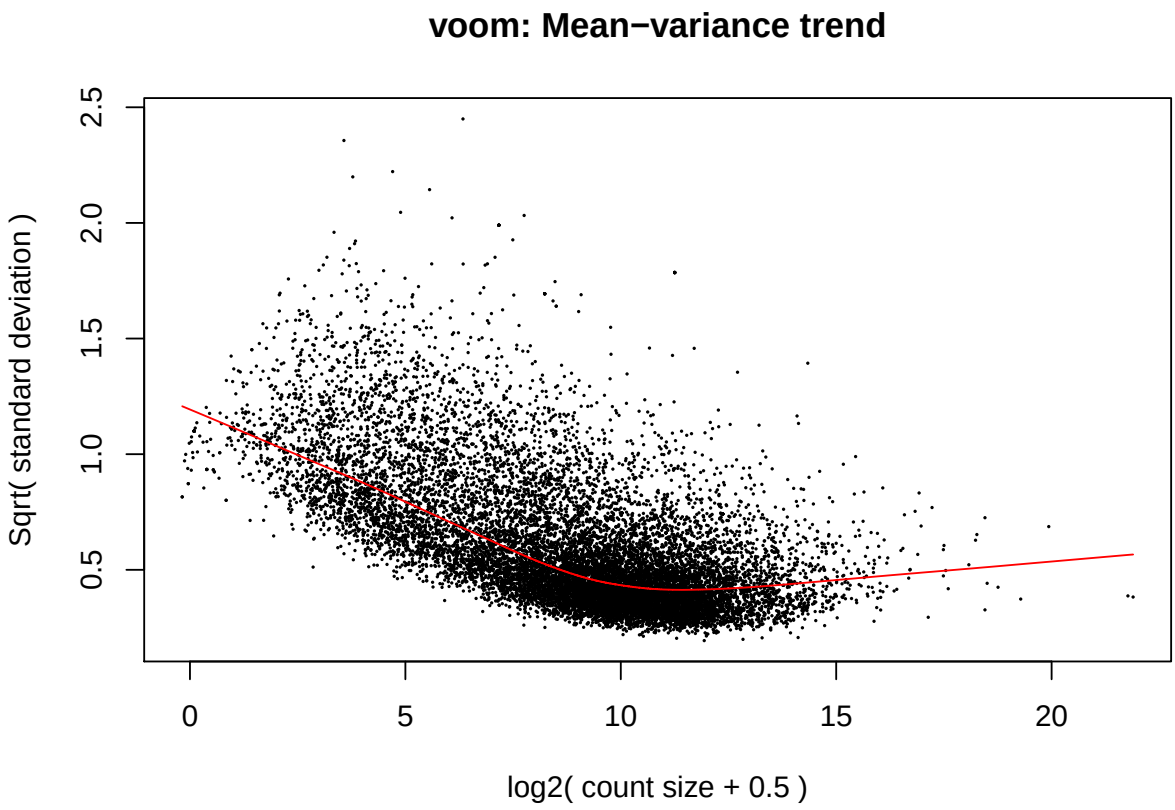
No quality weights, duplicate correlation or SVA analysis.

runVoom requires an SLOA object produced by runEdgeRNorm and containing count data (a DGElist) and a DesignMatrix.

Quality weights is enabled by default so we need to turn this off for the simple voom analysis. duplicateCorrelation is off by default so no argument is needed for this.

Code Block: Simple voom

```
MySLOA_simple = runVoom (MySLOA, PlotFile="MVplot_basic.png",
                  qualityWeights = FALSE)
saveRDS (MySLOA_simple, file = paste(ModelName, "_SLOA_simple.RDS", sep=""))
```



Fit data saved in DT_Model_SLOA_simple.RDS.

6.3 Voom with Quality Weights

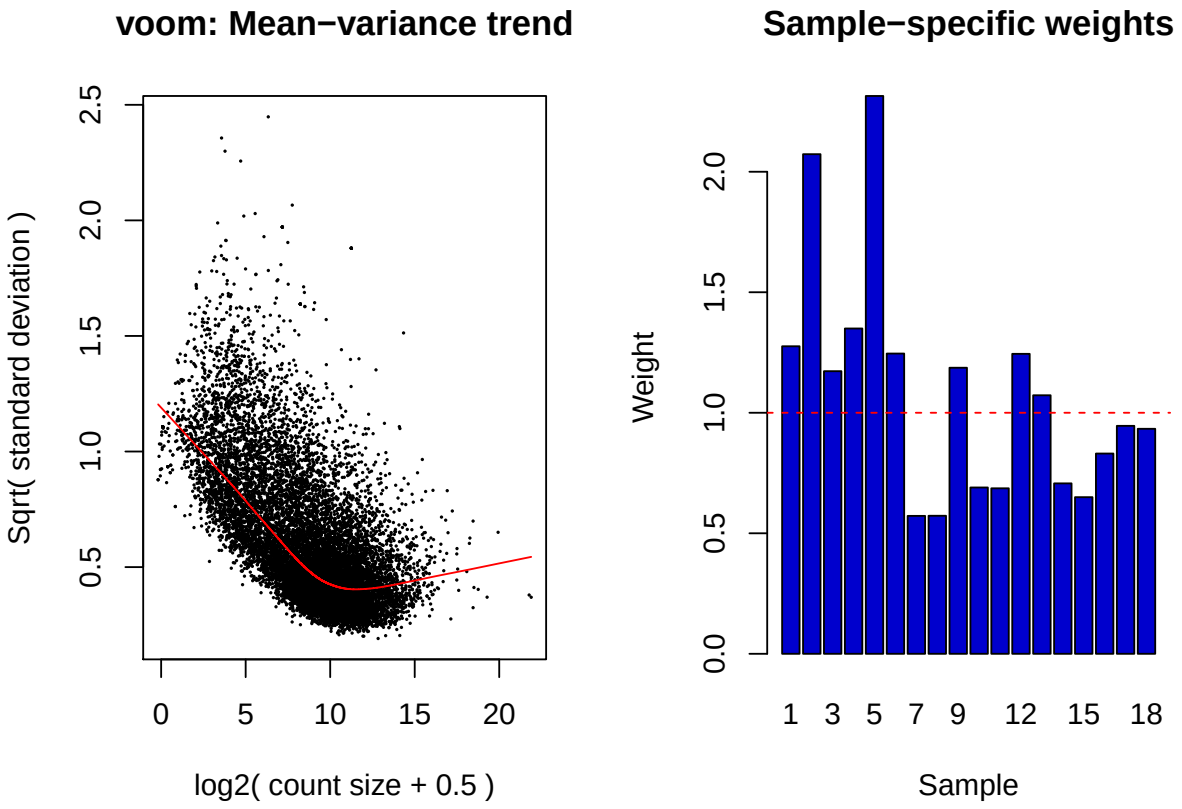
Quality weights are implmented by running voomWithQualityWeights instead of voom.

Quality weights are calculated based on the assumption that residuals from good data should be independent and identically distributed. Weights are assigned based on how well each sample conforms to these assumptions.

The use of quality weights was set as the default behavior. If all samples are of equal quality, there will be little benefit but no detriment either. But if all the quality weights are close to 1.0, you can confidently run without quality weights.

Code Block: voom with quality weights

```
MySLOA_qw = runVoom (MySLOA, PlotFile="MVplot_qw.png")
saveRDS (MySLOA_qw, file = paste(ModelName, "_SLOA_qw.RDS", sep=""))
```



Fit data saved in DT_Model_SLOA_qw.RDS.

Note the Quality Weights barplot shows that the samples differed in quality and therefore we'll keep the use of quality weights throughout the rest of these analyses.

6.4 Voom with Quality Weights and var.design

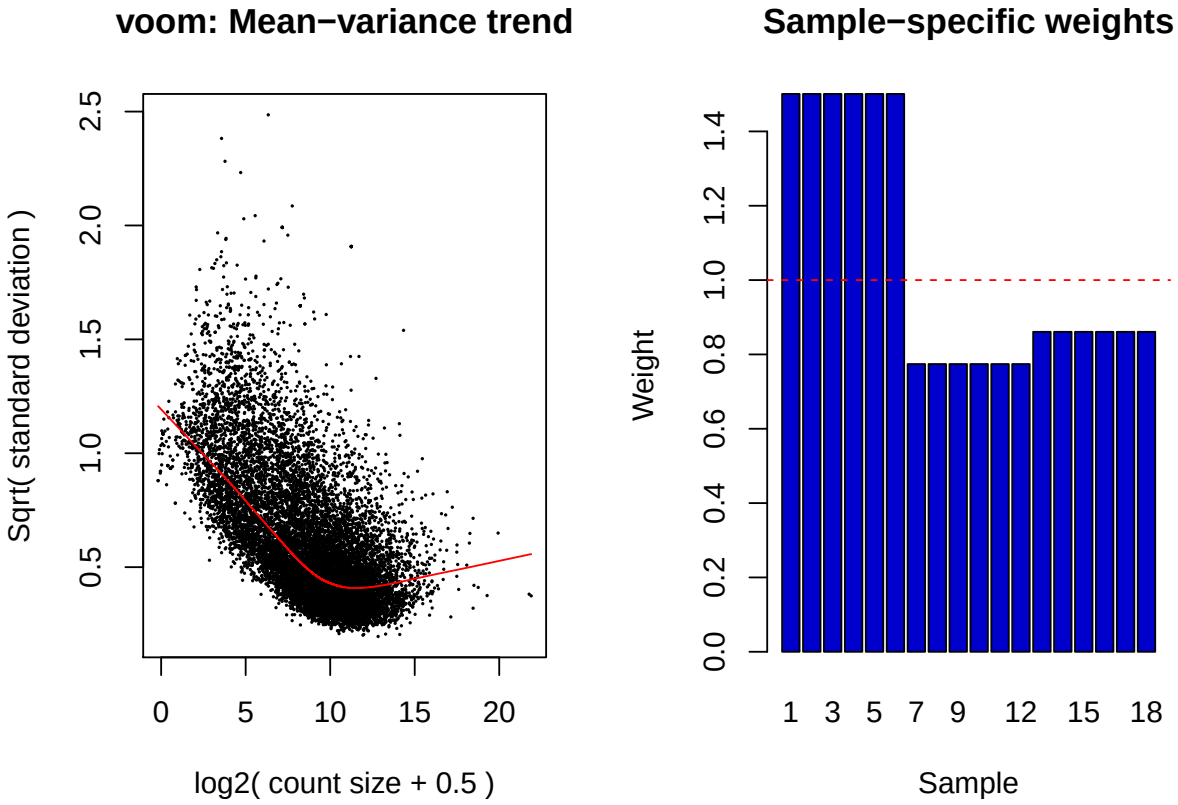
By default, without a var.design table, quality weights are determined for each sample individually. However, if groups of samples have similar quality, we can use that information to improve the quality weight assessment. With var.design, a quality weight is calculated for each group defined by var.design.

Notice in the barplot above, we see that the first six samples generally have a higher weight than the last 12 samples. The first six samples correspond to all the samples with DiseaseStatus = Normal. Thus we'll repeat the quality weight analysis using a var.design based on DiseaseStatus.

var.design will be used automatically if a var.design table is present in the SLOA. Thus, we'll simply create SLOA$var.design and re-run voomWithQualityWeights.

Code Block: Quality Weights with var.design

```
var.formula <- "~ 0 + Disease.Status"
MySLOA$var.design <- model.matrix(as.formula(var.formula), MySLOA_qw$colData)

MySLOA_qwvd <- runVoom (MySLOA, PlotFile="MVplot_qwvd.png")
saveRDS (MySLOA_qwvd, file = paste(ModelName, "_SLOA_qwvd.RDS", sep=""))
```



Fit data saved in DT_Model_SLOA_qwvd.RDS.

6.5 Voom with duplicateCorrelation

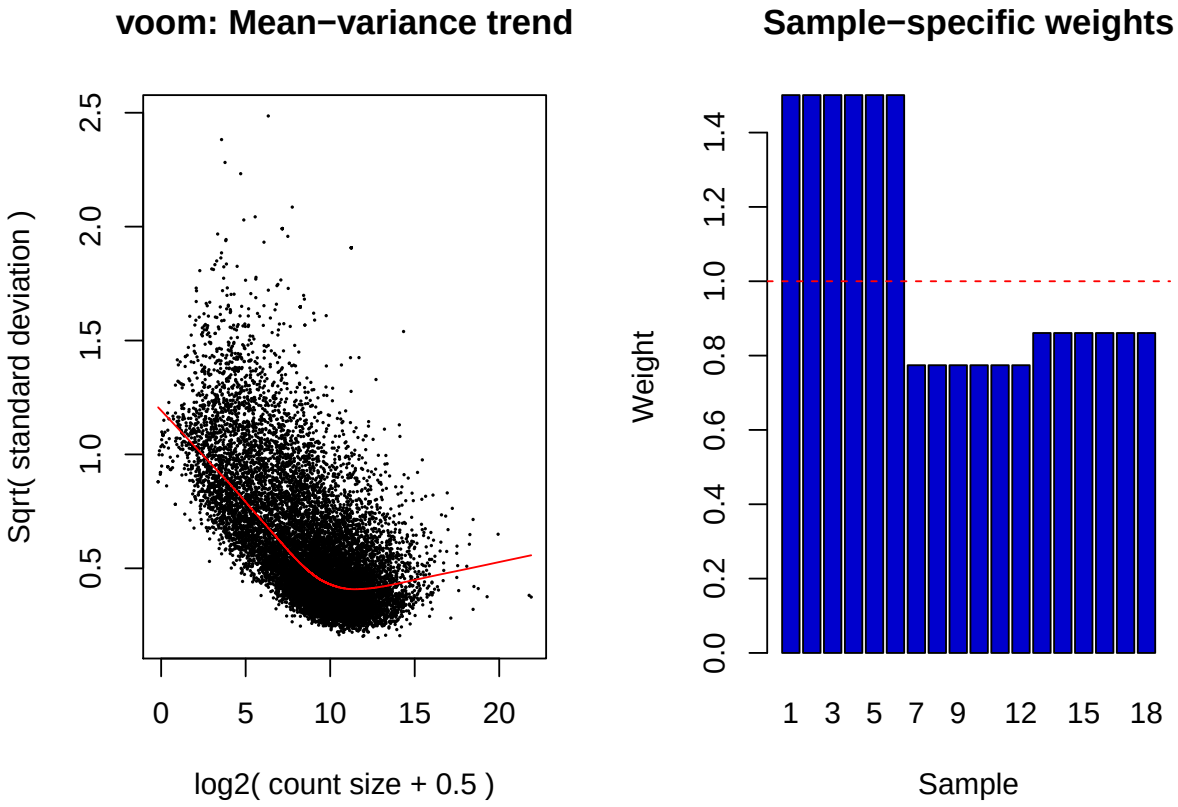
The duplicateCorrelation method is useful with nested designs. In this experiment each replicate represents a separate donor (9 donors total, 3 for each disease status). Each donor was tested both with and without TGFb treatment. Thus treatment is nested within donor. So we set up a block column to indicate which samples have the same donor.

Ideally, you'll consider the possible use of duplicateCorrelation in advance and include a column called "block" in your Design table when you build your RSE. In this case, we have to add the "block" column posthoc before running this analysis.

Note that were using MySLOA which now contains var.design and quality weights are enabled. So we're using duplicateCorrelation on top of the blocked quality weights for this analysis.

Code Block: voom with duplicateCorrelation

```
#Add a "block" column to the design data table which is stored in the SLOA as
#SLOA$colData.  Then  runVoom with dupCor=TRUE

#For this dataset, I can parse block info from the SampleName
sp = stri_split_regex(Design$SampleName, "[_-]")
#add a "block" column to the Design (colData)
MySLOA$colData$block = sapply(sp, "[[", 6)
#OK now the "block" column is in place.  Let's run the duplicateCorrelation
#analysis.

MySLOA_dupCor = runVoom (MySLOA, PlotFile="MVplot_dupCor.png", dupCor=TRUE)
saveRDS (MySLOA_dupCor, file = paste(ModelName, "_SLOA_dupCor.RDS", sep=""))
```



Fit data saved in DT_Model_SLOA_dupCor.RDS.

6.6 Voom Summary

We've run voom several ways, saving each run in a different SLOA and saving each SLOA in a different file. We could now run contrasts on each of those different SLOA objects. However, the quality weights barplot shows that quality weights are useful so we'll skip doing anything further with the simple data treatment.

Let's run contrasts with and without duplcateCorrelation and evaluate the effect of using duplicate correlation with pvalue histogram plots.

7 Run Contrasts

OK, we've decided to run contrasts with and without duplicateCorrelation to evaluate whether or not duplicateCorrelation was helpful.

By default, runContrasts run the eBayes method with robust=TRUE, followed by the topTable method to determine gene signatures with the defined contrasts for the null hypothesis that no genes are significantly differential. You can optionally turn on a topTreat analysis with the null hypothesis that no genes are differentially regulated at a specified fold-change level (Default = 1.5X).

To run contrasts, we set up a **Named List** of the different contrasts we want to perform. The contrasts are defined using the column names from the DesignMatrix. The names we give to each contrast are a short pneumonic that will be used in as labels in plots or heatmaps later.

See the design matrix for colnames available for constructing contrasts from your model.

```
> colnames(MySLOA$DesignMatrix)
[1] "DTNormalcontrol" "DTNormalTGFb"    "DTRapidcontrol"
[4] "DTRapidTGFb"     "DTStablecontrol" "DTStableTGFb"
```

Perform the following contrasts and get a signature for each of the below

Cell-Type Signatures, Untreated:

Compare the different cell types at baseline

- DTRapidcontrol - DTNormalcontrol
- DTStablecontrol - DTNormalcontrol
- DTRapidcontrol - DTStablecontrol

TGFb Signatures:

Compare the TGFb signatures vs untreated

- DTNormalTGFb - DTNormalcontrol
- DTRapidTGFb - DTRapidcontrol
- DTStableTGFb - DTStablecontrol

Cell-Type Signatures, TGFb treated:

Compare the cell types under TGFB stimulation

- (DTRapidTGFb-DTRapidcontrol)-(DTNormalTGFb-DTNormalcontrol)
- (DTStableTGFb-DTStablecontrol)-(DTNormalTGFb-DTNormalcontrol)
- (DTRapidTGFb-DTRapidcontrol)-(DTStableTGFb-DTStablecontrol)

7.1 Signature Genes Tables

For each contrast analysis, we've calculated pvalues for 2 null hypotheses:

- Change = 0 (using topTable)
- abs(Change) <= 1.5FC (using topTreat)

Run the contrasts both with and without the duplicateCorrelation.

Contasts saved to:
DT_Model_qwvd_Contrasts.RDS
DT_Model_dupCor_Contrasts.RDS

The Signature Gene Table tabulate the differential gene count for various thresholds both with and without the duplicateCorrelation piece.

Code Block: Contrasts for Model with blocked qualityWeights, Without duplicateCorrelation

```
#Define the contrasts of interest:

MyContrasts = list(RapidVsNorm = "DTRapidcontrol - DTNormalcontrol",
              StableVsNorm = "DTStablecontrol - DTNormalcontrol",
              RapidVsStable = "DTRapidcontrol - DTStablecontrol",
              TGFb_Norm = "DTNormalTGFb - DTNormalcontrol",
              TGFb_Rapid = "DTRapidTGFb - DTRapidcontrol",
              TGFb_Stable = "DTStableTGFb - DTStablecontrol",
              TGFb_RvsTGFb_N =
                "(DTRapidTGFb-DTRapidcontrol)-(DTNormalTGFb-DTNormalcontrol)",
              TGFb_SvsTGFb_N =
                "(DTStableTGFb-DTStablecontrol)-(DTNormalTGFb-DTNormalcontrol)",
              TGFb_RvsTGFb_S =
                "(DTRapidTGFb-DTRapidcontrol)-(DTStableTGFb-DTStablecontrol)"
)

DTcontrasts = runContrasts(MySLOA_qwvd, MyContrasts, runTopTreat=TRUE,
                           SigTablePNG="SigTable_qwvd.PNG")
saveRDS(DTcontrasts, file = paste(ModelName, "_qwvd_Contrasts.RDS", sep=""))
```

	P<0.01	10% FDR	10% LFDR	P<0.01 & 1.5FC	10% FDR & 1.5FC
RapidVsNorm	1679	1893	1555	319	1
StableVsNorm	1287	1104	981	181	0
RapidVsStable	188	0	0	25	0
TGFb_Norm	4782	6631	5791	1108	997
TGFb_Rapid	4192	5916	4957	1168	1060
TGFb_Stable	3464	4905	4064	789	582
TGFb_RvsTGFb_N	122	0	0	16	0
TGFb_SvsTGFb_N	26	0	0	1	0
TGFb_RvsTGFb_S	10	0	0	3	0

Code Block: Contrasts for Model Including duplicateCorrelation

```
DTcontrasts_dupCor = runContrasts(MySLOA_dupCor, MyContrasts, runTopTreat=TRUE,
                                  SigTablePNG="SigTable_dupCor.PNG")
saveRDS(DTcontrasts_dupCor, file = paste(ModelName, "_dupCor_Contrasts.RDS", sep=""))
```

	P<0.01	10% FDR	10% LFDR	P<0.01 & 1.5FC	10% FDR & 1.5FC
RapidVsNorm	1642	1826	1468	342	2
StableVsNorm	1233	1059	926	216	0
RapidVsStable	185	0	0	35	0
TGFb_Norm	7085	9212	8458	1716	1801
TGFb_Rapid	6490	8650	7774	1941	2137
TGFb_Stable	5761	7831	6954	1375	1354
TGFb_RvsTGFb_N	620	6	31	33	1
TGFb_SvsTGFb_N	239	0	0	9	0
TGFb_RvsTGFb_S	102	0	0	7	0

Note that, comparing the two signature tables, running duplicateCorrelation increased sensitivity primarily on the TGFb treatments.

7.2 PValue Histograms

A pvalue histogram where the null hypothesis is true (no differential gene expression) should show evenly distributed pvalues between 0 and 1. If some genes are differentially expressed, a peak at low pvalues will emerge. Any other behavior on these plots indicates a problem with the statistical model.

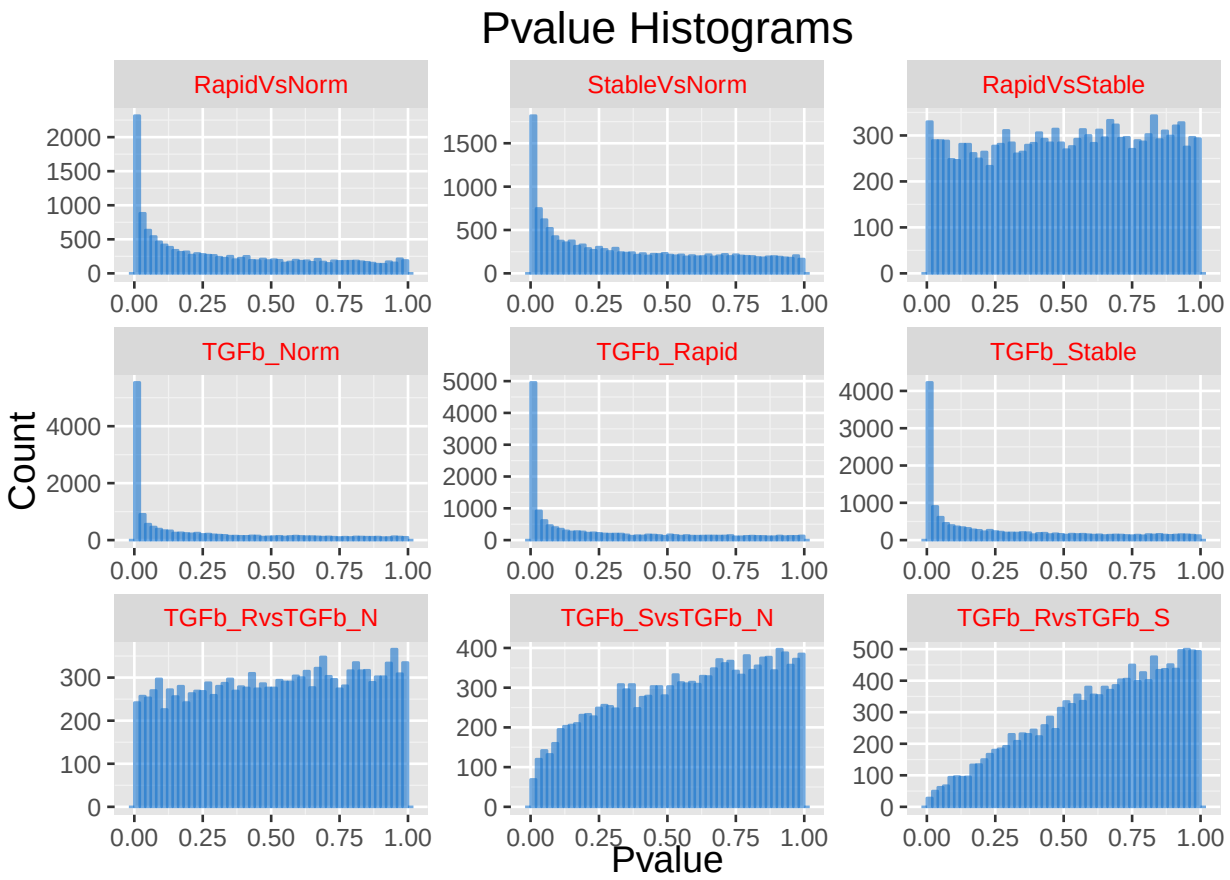
Here we'll evaluate the pvalue histograms of the contrasts with and without the duplicateCorrelation analysis.

This webpage provides a nice explaination of what several non-ideal pvalue histograms might mean.

7.2.1 Before using duplicateCorrelation

Code Block: Pvalue Histograms Before duplicateCorrelation

```
#Pvalue histograms

#create a dataframe of Pvalues for each contrast
Pval <- extractCol(DTcontrasts$TopTableList, "P.Value")

#facet plot
pvalPlot <- plotPvalHist(Pval, facetFontSize = 10, savePlot=F)
printAndSave(pvalPlot, filename="PvalHist.PNG", printFontSize=10, saveFontSize=14)
```



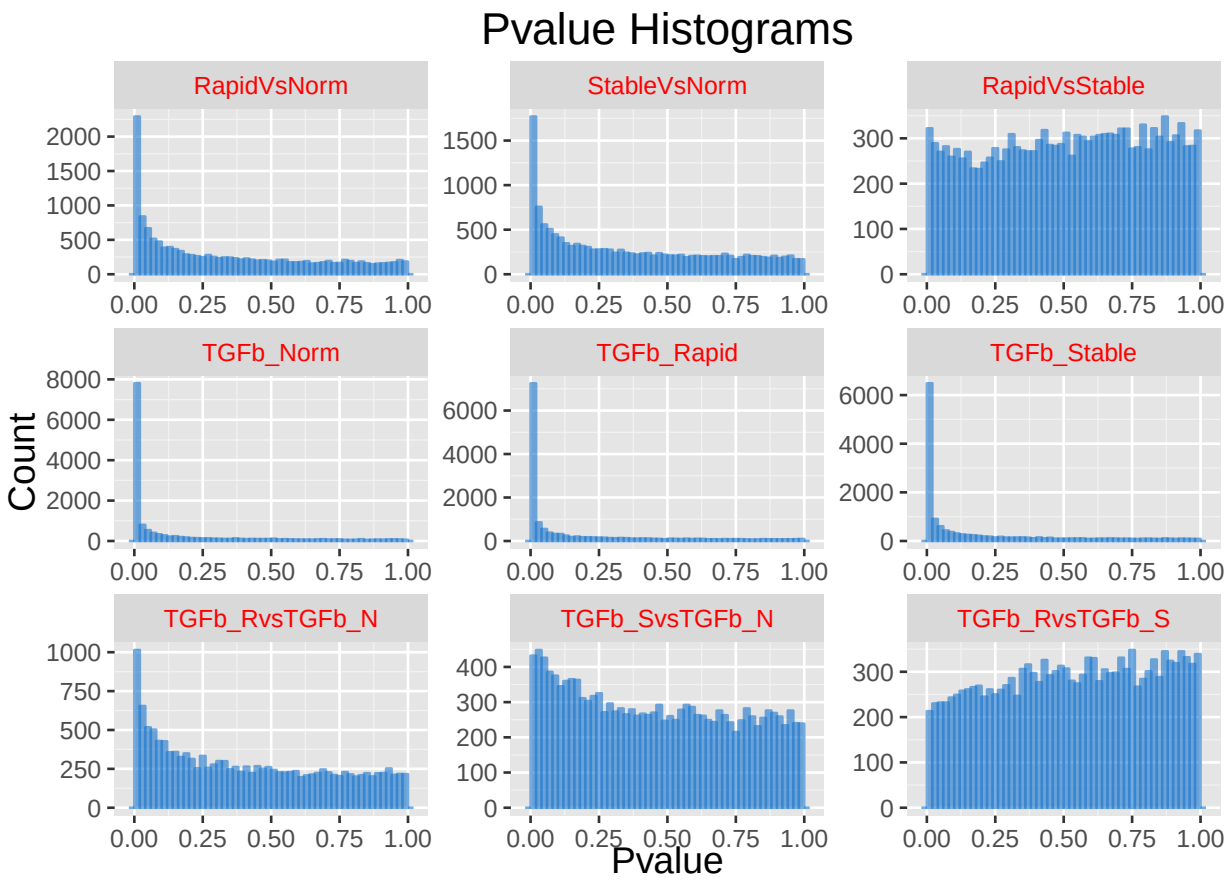
The first two rows show a flat distribution between 0-1 with a peak at low pvalues indicative of a differential gene signature. The lack of a low pvalue peak in the Rapid vs. Stable contrast, indicates there is essentially no significant gene signature there.

The bottom row shows an undesirable bias toward high pvalues in the distribution. Pay attention to this last row after we apply duplicateCorrelation.

7.2.2 After using duplicateCorrelation

Code Block: Pvalue Histograms After duplicateCorrelation

```
#Pvalue histograms
Pval <- extractCol(DTcontrasts_dupCor$TopTableList, "P.Value")
#print for the report
pvalPlot <- plotPvalHist(Pval, facetFontSize = 10, savePlot=F)
printAndSave(pvalPlot, filename="PvalHist_dupcor.PNG", printFontSize=10, saveFontSize=14)
```



Observations:

The undesirable high pvalue bias in the third row is now corrected (flattened) indicating the model including a duplicateCorrelation method best fits this data. After applying duplicate correlation, weak signatures are now evident in the first two plots on row three, thus illustrating that the correct model improves your ability to detect differentially expressed genes.

8 MDS : Multi-Dimensional Scaling

Multi-Dimensional Scaling (MDS) is a data reduction method that reduces a large data set to principal components. MDS is a generalization of PCA. In other words, PCA is a special case of MDS. PCA is running MDS on all genes with a euclidean distance metric. MDS generalizes the method to other distance metrics and subsets of genes.

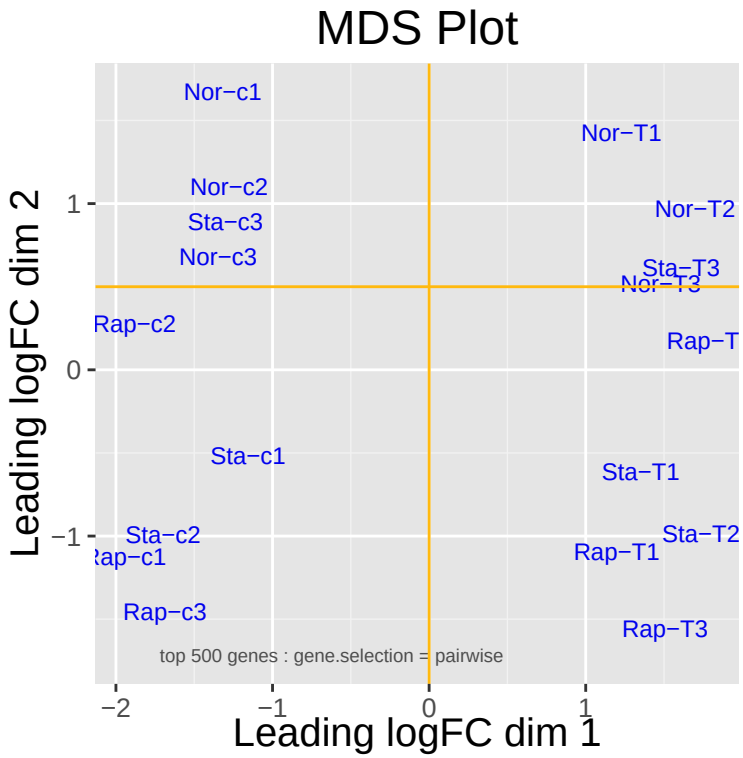
On a good day, the biological parameters of your data should map to one or more principal components. In this dataset the two main biological parameters are treatment (Vehicle or TGFb treatment) and cell type (Norm, Rapid, Stable).

The MDS distance metric is tailored to expression data. It uses the top 500 genes by Default and a distance metric using the logFC difference between pairs. The number of genes evaluated can be changed by the "top" argument. In this dataset, TGFb regulates >5000 genes. Thus, the plotMDS default of 500 is only using a small subset of all regulated genes and we may consider increasing the value over the default.

Let's try the default top = 500 first. Then we'll repeat with top = 5000. We'll use the duplicateCorrelation data treatment for this analysis. The DGElist object is the proper input to the MDS method. The DGElist is found in the SLOA object.

Code Block: MDS with Top 500 (plotMDS default)

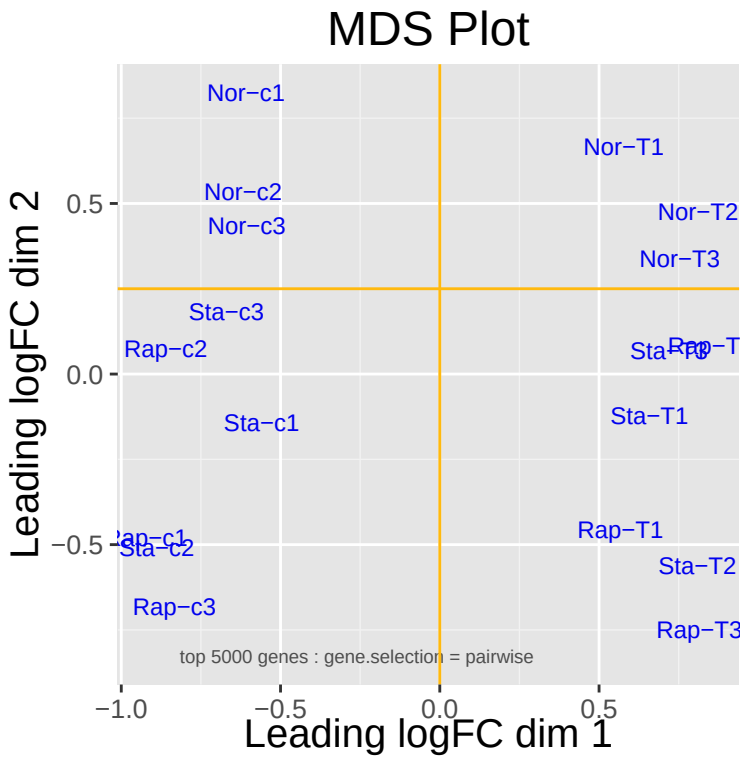
```
#get some labels for the plot
Treatment <- DTcontrasts_dupCor$colData$Treatment
DiseaseStatus <- DTcontrasts_dupCor$colData$Disease.Status
#construct a short label: 1st 3 letters of DiseaseStatus, 1st letter of
#Treatment and a rep number
Label = paste (str_sub(DiseaseStatus,1,3), "-", str_sub(Treatment,1,1),
               c(1:3), sep="")

DGElist = MySLOA_dupCor$DGElist
MDS = ggplotMDS(DGElist, labels= Label, labelSize=3, hlineIntercept = 0.5,
                vlineIntercept = 0, gene.selection="common", top=500)
printAndSave(MDS$plot, file="MDSplot500.PNG")
```



Notice that instead of running plotMDS directly, I've used a wrapper function (ggplotMDS) that uses ggplot to give you more flexibility on the appearance of the plot. ggplotMDS takes the same arguments as plotMDS and adds more options to control the graphic appearance. Use ?ggplotMDS to see the options. Notably, I've changed the default top value to Inf (all genes) for reasons elucidated below. Moreover, ggplotMDS returns a list containing both the ggplot object (MDS$plot) and the mdsobj returned by plotMDS (MDS$mdsobj). The mdsobj contains the underlying MDS data that can be used for other analyses (e.g. correlate MDS values with biological data).

Code Block: MDS with TOP 5000

```
MDS = ggplotMDS(DGElist, labels= Label, labelSize=3, hlineIntercept = 0.25,
                vlineIntercept = 0, gene.selection="common", top=5000)
printAndSave(MDS$plot, file="MDSplot5000.PNG")
```



Observations

Dim1 clearly separates samples based on TGFb treatment. On the other hand, Dim2 separates Normal from Stable & Rapid. Note that the separation on Dim2 is greater with top=5000. Thus it seems reasonable to pick the value for top to be close to your gene signature size. Going up to top=Inf, the plot still looks the same (not shown) although the scale changes (because you're including more low fold change genes). Thus, in practical terms, you should tweak the value of top in each situation to convince yourself the result is stable.

9 SVA Analysis : Removing unknown confounding factors

If you think there are still unknown factors or variables in an experiment, e.g. the first principal component in an MDS or PCA analysis does not track with known biology or experimental factors or your pvalue plots don't look right. Then we can use SVA analysis to identify and correct for unknown factors in a dataset.

Operationally, you run the SVA analysis after runVoom, then re-run runVoom with the new SLOA object created by runSVA. The runSVA function identifies surrogate variables and adds them to the Design table and DesignMatrix such that runVoom now incorporates the surrogate factors into the statistical experiment design and fit.

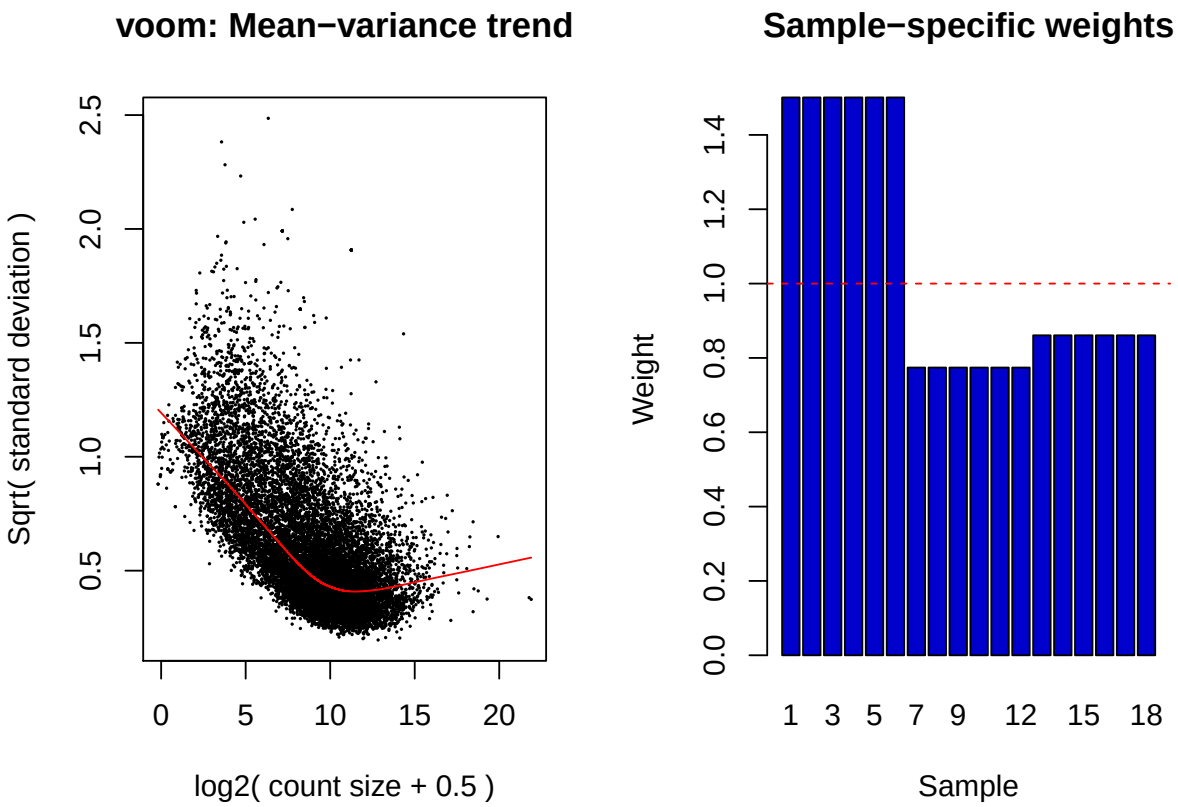
Here, we'll run SVA analysis on the SLOA object derived from runVoom with quality weights and duplicateCorrleation. Then we'll repeat the runVoom step, again with quality weights and duplicateCorrelation enabled.

Code Block: runSVA followed by runVoom and runContrasts

```
MySLOA_sva <- runSVA(MySLOA_dupCor)
```

```
## Number of significant surrogate variables is:  5
## Iteration (out of 5 ):1  2  3  4  5
```

```
#now repeat the voom/lmFit analysis
MySLOA_sva = runVoom (MySLOA, PlotFile="MVplot_sva.png", dupCor=TRUE)
```



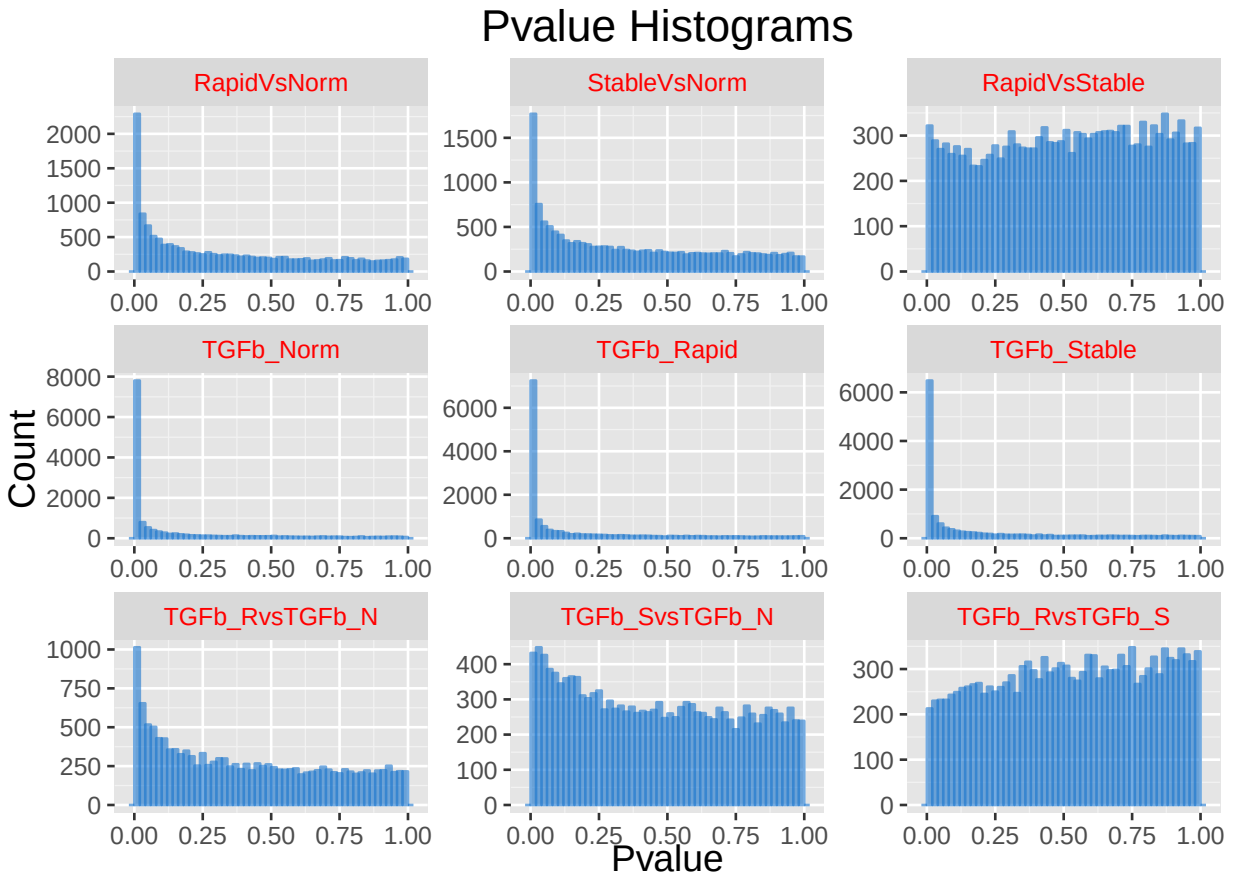
```
saveRDS (MySLOA_sva, file = paste(ModelName, "_SLOA_sva.RDS", sep=""))

#And finally repeat the contrast analysis
DTcontrasts_sva = runContrasts(MySLOA_sva, MyContrasts, runTopTreat=TRUE,
                                SigTablePNG="SigTable_sva.PNG")
```

	P<0.01	10% FDR	10% LFDR	P<0.01 & 1.5FC	10% FDR & 1.5FC
RapidVsNorm	1642	1826	1468	342	2
StableVsNorm	1233	1059	926	216	0
RapidVsStable	185	0	0	35	0
TGFb_Norm	7085	9212	8458	1716	1801
TGFb_Rapid	6490	8650	7774	1941	2137
TGFb_Stable	5761	7831	6954	1375	1354
TGFb_RvsTGFb_N	620	6	31	33	1
TGFb_SvsTGFb_N	239	0	0	9	0
TGFb_RvsTGFb_S	102	0	0	7	0

```
saveRDS(DTcontrasts_sva, file = paste(ModelName, "_sva_Contrasts.RDS", sep=""))
```

Code Block: pvalue histograms on SVA data treatment

```
#Pvalue histograms
Pval <- extractCol(DTcontrasts_sva$TopTableList, "P.Value")
#print for the report
pvalPlot <- plotPvalHist(Pval, facetFontSize = 10, savePlot=F)
printAndSave(pvalPlot, filename="PvalHist_sva.PNG", printFontSize=10, saveFontSize=14)
```



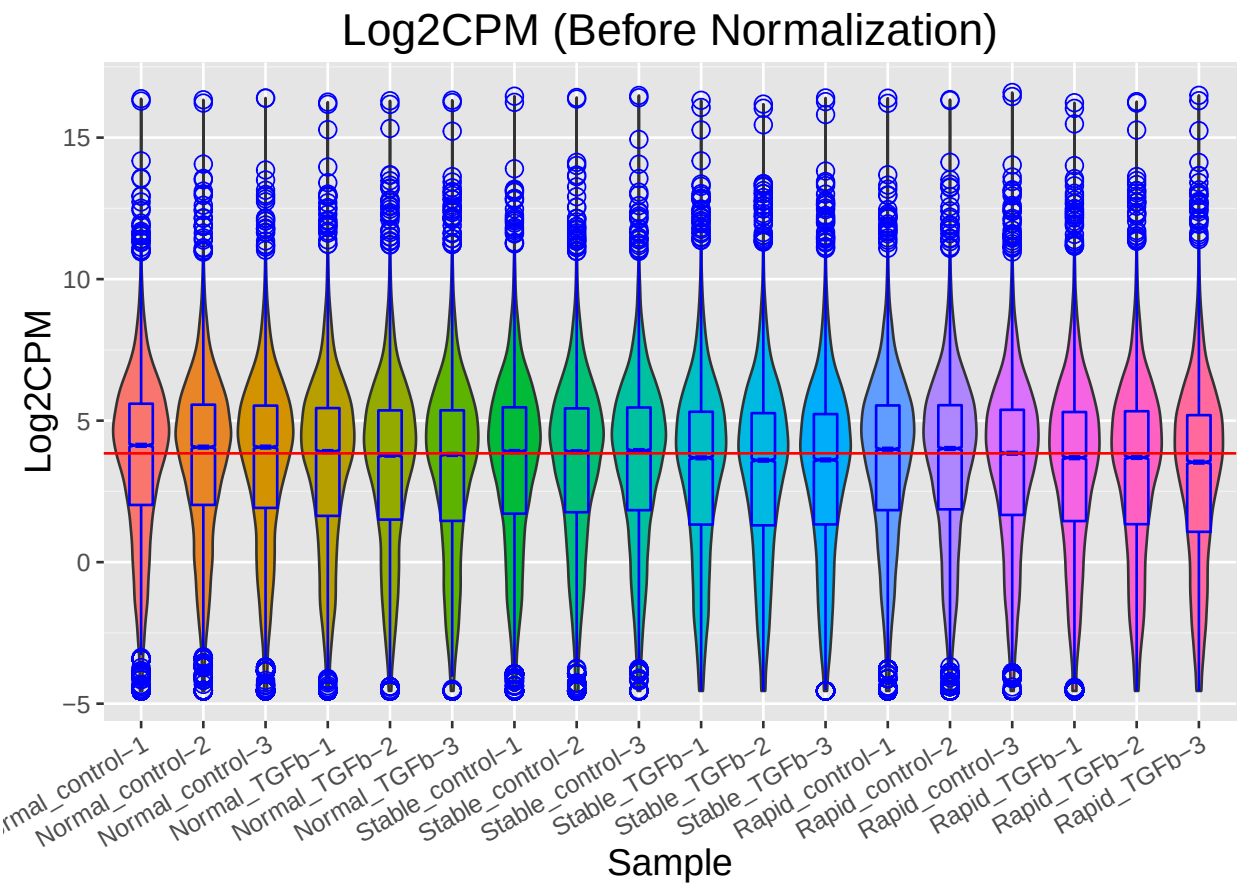
Note: In this dataset, even before we ran SVA, the first and second principal components correlated with the primary biology in the experiment (TGFb treatment and disease status). Thus, this dataset didn't appear to have significant hidden variables at the get-go. Confirming this, the pvalue plots show no benefit in this particular case. So this illustrates operationally how to run SVA, but this dataset didn't need it.

10 Evaluating TMM Normalization

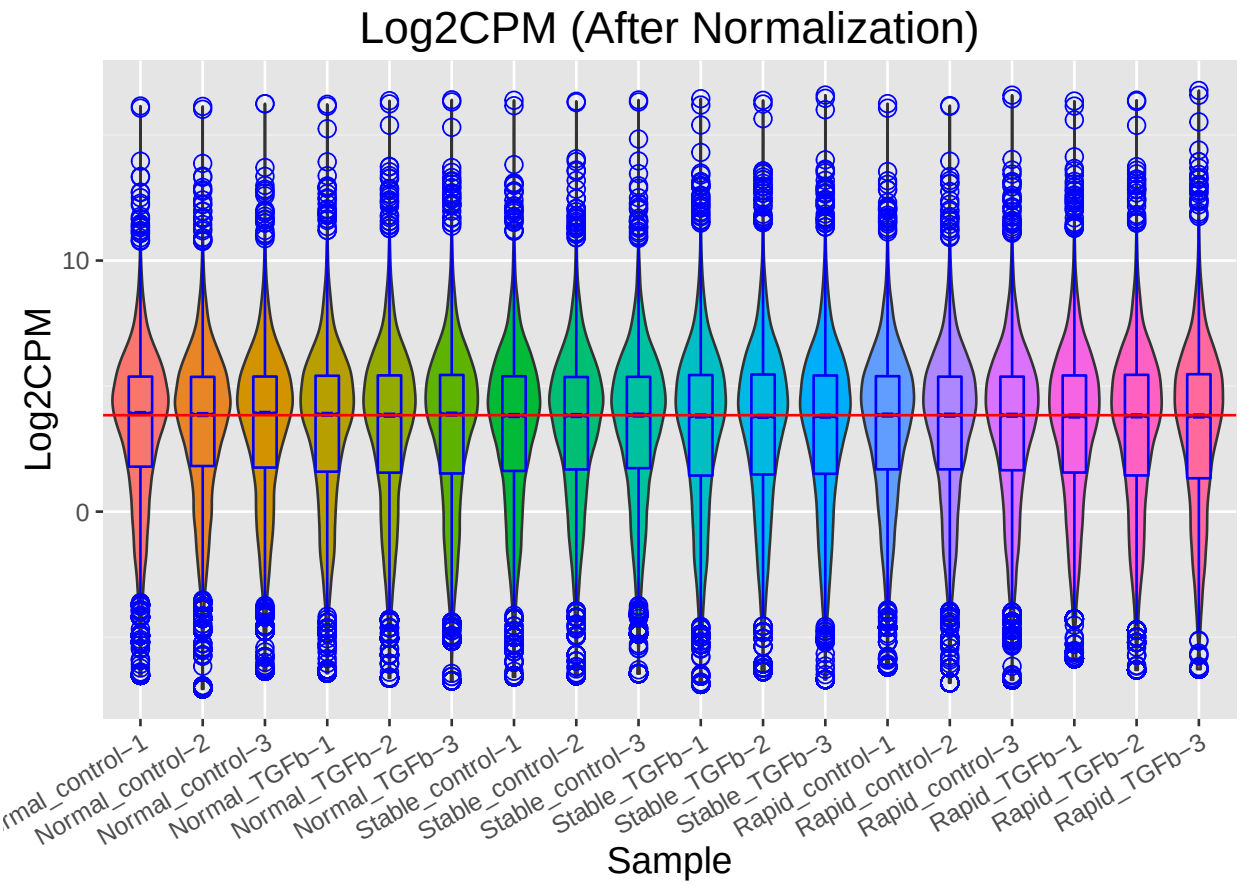
EdgeR normalization normalized the count data. After running voom, we have normalize Log2CPM at our disposal. This comparison of log2CPM values before and after EdgeR TMM normalization is useful to view the effects of normalization. The RawLog2CPM values are calculated from the original, un-normalized counts using the cpm function from the edgeR package.

Plot a boxplot of log2CPM values before and after normalization to observe the benefits of normalization.

```
#configure data for plotting
preCPM = assay(RSE, "RawLog2CPM")
colnames(preCPM) = paste(Design$ReplicateGroup, 1:3, sep="-")
premedian = median(preCPM)

postCPM = MySLOA_dupCor$Log2CPM
colnames(postCPM) = paste(Design$ReplicateGroup, 1:3, sep="-")
postmedian = median(postCPM)

#now draw the boxplots

#Before Normalization
m1 = melt(as.data.frame(preCPM))
preCPMplot = ggplot(m1, aes(x = variable, y = value)) +
  geom_violin(aes(fill = factor(variable))) +
  geom_boxplot(color = "blue", notch = T, outlier.shape=1, outlier.colour = "blue", width=0.5,
               outlier.size=3, aes(fill = factor(variable))) +
  geom_hline(yintercept=premedian, color="red") +
  ggtitle ("Log2CPM (Before Normalization)") +
  guides(fill=guide_legend(title=NULL)) +
  guides(fill=FALSE) +
  xlab("Sample") +
  ylab("Log2CPM") +
  greyTheme(12) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1))

printAndSave(preCPMplot, filename="preCPMboxplot.PNG")
```




```
#After Normalization
m2 = melt(as.data.frame(postCPM))
postCPMplot =ggplot(m2,aes(x = variable,y = value)) +
  geom_violin(aes(fill = factor(variable))) +
  geom_boxplot(color = "blue", notch = T, outlier.shape=1, outlier.colour = "blue", width=0.5,
               outlier.size=3, aes(fill = factor(variable))) +
  geom_hline(yintercept=postmedian, color="red") +
  ggtitle ("Log2CPM (After Normalization)") +
  guides(fill=guide_legend(title=NULL)) +
  guides(fill=FALSE) +
  xlab("Sample") +
  ylab("Log2CPM") +
  greyTheme(12) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1))

printAndSave(postCPMplot, filename="postCPMboxplot.PNG")
```



A horizontal line is drawn at the median of all samples.

The difference is subtle with this dataset, indicating the data was good quality to begin with. However, Samples 1-3 are slightly above the median in the before plot. Similarly, samples 10-12 are slightly below median in the before plot. In the after plot, all samples are nicely aligned with the median line.

The after normalization data has been TMM normalized and voom squeezed and is the same log2CPM values submitted to lmFit.

11 Session Info

Time required to process this report: *4.662876 mins*

R Session Info

Section	Name	Value
System	sysname	Windows
System	release	7
System	version	build 7601, Service Pack 1
System	nodename	PF029MHB
System	machine	x86
System	login	thompj27
System	user	thompj27
System	effective_user	thompj27
System	Directory	C:/Users/thompj27/Documents/Fibrosis/IPF Cedar Sinai/RNA-Seq/RefGene2015/R
R	Version	R version 3.2.3 (2015-12-10)
Packages	knitr	1.11 CRAN CRAN 2015-08-14
Packages	envDocument	2.1.1 BMS BMS 2015-10-24
Packages	gridExtra	2.0.0 CRAN CRAN 2015-07-14 20:40:38
Packages	reshape2	1.4.1 CRAN CRAN 2014-12-06 06:56:59
Packages	stringr	1.0.0 CRAN CRAN 2015-04-30 11:48:24
Packages	stringi	1.0-1 CRAN CRAN 2015-10-22
Packages	DGE.Tools	1.0 NA NA NA
Packages	SummarizedExperiment	1.0.1 NA NA NA
Packages	Biobase	2.30.0 NA NA NA
Packages	GenomicRanges	1.22.1 NA NA NA
Packages	GenomeInfoDb	1.6.1 NA NA NA
Packages	IRanges	2.4.2 NA NA NA
Packages	S4Vectors	0.8.3 NA NA NA
Packages	BiocGenerics	0.16.1 NA NA NA
Packages	ggplot2	2.0.0 CRAN CRAN 2015-12-18 10:45:15
Packages	dplyr	0.4.3 CRAN CRAN 2015-09-01 18:15:02
Packages	magrittr	1.5 CRAN CRAN 2014-11-22 19:15:57
Packages	BiocInstaller	1.20.1 NA NA NA
Script	Path	C:/Users/thompj27/Documents/Fibrosis/IPF Cedar Sinai/RNA-Seq/RefGene2015/R
Script	Modified	2015-12-30 00:05:55