Further Analysis of RNA-seq data

Mark Dunning

Last modified: March 14, 2016

Contents

1 Clustering and PCA

1.1 Introduction to the DESeq2 Vignette

The general application of clustering to RNA-seq data is outlined by Section 2.2 of the *DESeq2* vignette. We will follow this section of the vignette closely.

The DESeq2 vignette uses the *pasilla* dataset; a popular example dataset in Bioconductor concerning the RNAi knock-out of the pasilla splicing factor in Drosophilla. Full details of the dataset can be found in the vignette for the *pasilla* package. For convenience, the gene-level counts for these data are available as the `pasillaGenes` object in the *pasilla* package.

The counts and experiment metadata can be accessed using the `counts` and `pData` functions respectively.

1.2 Importing the pasilla dataset

Use Case: Load the pasilla Bioconductor package and save the counts and metadata as new R objects.

```
library("pasilla")
library("Biobase")
data("pasillaGenes")
countData <- counts(pasillaGenes)
head(countData)
pData(pasillaGenes)
colData <- pData(pasillaGenes)[,c("condition","type")]
```

A dataset ready for analysis with DESeq2 can be constructed using the `DESeqDataSetFromMatrix` function in DESeq2. You need to specify the counts matrix and experiment metadata, along with an experimental design. In this case, we want to test for *treated* versus *untreated*.

Use Case: Create a DESeq2 dataset for the pasilla dataset

```
library(DESeq2)
dds <- DESeqDataSetFromMatrix(countData = countData,
  colData = colData,
  design = ~ condition)
dds
```

In some of the analysis we are going to perform, it will be useful to supply extra annotation for the features (genes). The DESeq2 package makes extensive use of the GenomicRanges infrastructure that we introduced previously in the course. Specifically, we can add feature annotation to the `mcols` of the dataset

Use Case: Add feature annotation to the dataset

```
featureData <- data.frame(gene=rownames(pasillaGenes))
(mcols(dds) <- DataFrame(mcols(dds), featureData))
```

Use Case: Calculate the scaling factors for the dataset

```
dds <- estimateSizeFactors(dds)
```

1.3 Quality assessment using heatmaps, clustering and PCA

Data quality assessment and quality control (i. e. the removal of insufficiently good data) are essential steps of any data analysis. These steps should typically be performed very early in the analysis of a new data set, preceding or in parallel to the differential expression testing. We define the term quality as fitness for purpose. Our purpose is the detection of differentially expressed genes, and we are looking in particular for samples whose experimental treatment suffered from an anormality that renders the data points obtained from these particular samples detrimental to our purpose.

To produce heatmaps, we will use the *pheatmap* (*"pretty heatmap"*) package. Similar steps can be performed in the *heatmap.plus* package, the heatmap.2 function in *gplots* or the base heatmap function.

Drawing a heatmap can be computationally-expensive. Also, they are more-easily interpretable when the number of rows (genes) is low. Typically we filter our dataset to only include informative genes that are expressed or variable in our dataset.

Use Case: Select the 20 most-highly expressed genes in the dataset

```
library("pheatmap")
N <- 20
select <- order(rowMeans(counts(dds,normalized=TRUE)),decreasing=TRUE)[1:N]
```

In order to test for differential expression, we operate on raw counts and use discrete distributions as described in the previous practical. However for other downstream analyses – e.g. for visualization or clustering – it might be useful to work with transformed versions of the count data.

Maybe the most obvious choice of transformation is the logarithm. Since count values for a gene can be zero in some conditions (and non-zero in others), some advocate the use of *pseudocounts*, i. e. transformations of the form

$$y = \log_2(n+1) \quad \text{or more generally,} \quad y = \log_2(n+n_0), \tag{1}$$

where n represents the count values and n_0 is a positive constant. The normTransform

The *DESeq2* vignette goes on to describe the *regularized log* and *variance stabilizing* transformations, which might be of interest to more-advanced users.

Use Case: Transform the data onto a scale suitable for visualisation (e.g. $\log_2$). Extract the transformed values for the 20 most highly expressed genes

```
nt <- normTransform(dds) # defaults to log2(x+1)
log2.norm.counts <- assay(nt)[select,]
```

We are now ready to produce the heatmap. The function we are going to use is pheatmap. As usual, we can find out more about this function by typing ?pheatmap. It is useful to annotate the columns (samples) in the heatmap using particular levels from the experiment metadata. pheatmap is able to add this information provided we supply it with a data frame with the same number of *rows* as the number of columns in our matrix. We can have as many columns of meta data as we like.

Use Case: Create a data frame that has the condition and read-type (single or paired-end) for each sample. Produce a heatmap of the normalized counts of your selected genes that incorporates the metadata for each sample.

```
df <- as.data.frame(colData(dds)[,c("condition","type")])
pheatmap(log2.norm.counts, cluster_rows=FALSE, show_rownames=FALSE,
cluster_cols=FALSE, annotation_col=df)
```

comment: We have chosen not to cluster the rows or columns, so the columns in the plot appear in the same order as the original matrix. Sometimes we might want to decide this order by clustering, as *we will do in subsequent examples.*

Another use of the transformed data is sample clustering. This is done in an un-supervised manner to see if the clustering can uncover the known sample groups in our data.

We will switch our analysis to the *regularized log* transformed data, which can be computed using the `rlog` function. This transformation is described in detail in Section 2.1.3 of the DESeq2 vignette. The `dist` function can then be use to calculate pairwise distances between all samples. We have to remember to *transpose* the transformed matrix using the `t` function.

Use Case: Compute the regularized log intensities and use these to calculate a distance matrix. What metric is being used to calculate the distances?

```r
rld <- rlog(dds)
rlog.intensities <- assay(rld)
head(rlog.intensities)
sampleDists <- dist(t(rlog.intensities))
sampleDists
```

If we wanted, we could use the standard hierachical clustering function in R, `hclust`, to cluster the samples.

Use Case: Produce a dendrogram to visualise the clustering of the samples. Does it confirm the known sample groups?

```r
hc <- hclust(sampleDists)
plot(hc)
```

A more-appealing visualisation can be produced by `pheatmap`. In the previous example, we told `pheatmap` to retain the original column and row orders. Another way of using `pheatmap` is to supply a pre-computed distance matrix.

Use Case: Produce a heatmap to visualise the sample relationships from the distance matrix that you just computed.

```r
sampleDistMatrix <- as.matrix(sampleDists)
rownames(sampleDistMatrix) <- paste(rld$condition, rld$type, sep="-")
colnames(sampleDistMatrix) <- NULL


pheatmap(sampleDistMatrix,
clustering_distance_rows=sampleDists,
clustering_distance_cols=sampleDists)
```

Another way of customising the heatmap is to specify the colour palette used to colour each cell in the matrix. Many visually-appealing palettes are provided in the *RColorBrewer* package.

Use Case: Load the *RColorBrewer* package and see what palettes are available

```
library("RColorBrewer")
display.brewer.all()
```

The `colorRampPalette` function can be used to interpolate a set of given colours

Use Case: Create a palette that ranges from dark-blue (low) to white (high) and use this palette in the heatmap

```
colors <- colorRampPalette(rev(brewer.pal(9, "Blues")) )(255)
pheatmap(sampleDistMatrix,
clustering_distance_rows=sampleDists,
clustering_distance_cols=sampleDists,
col=colors)
```

comment: You should make sure that the colour palette you choose are suitable for those with colour-blindness

1.4 PCA

Principal Components Analysis (PCA) is a useful dimension-reduction technique that can highlight relationships between samples in our dataset. As it can be computationally-expensive, it is best performed on a filtered version of the data. Typically, we pick the "most-variable" genes. The `rowVars` function in *genefilter* is useful for this task. DESeq2 actually provides a function that automates the steps required to do a PCA analysis of a `DESeqDataSet` object and produce an informative plot. However, we will go through the steps manually to illustrate the method.

Use Case: Create a vector of indices for the 500 most variable genes according to the `rld` data

```
library(genefilter)
rv <- rowVars(assay(rld))
select <- order(rv, decreasing=TRUE)[1:500]
```

PCA can be performed using the `prcomp` function. As when computing a distance matrix, we need to remember to *transpose* our matrix of intensities if we want to do PCA on the samples.

Use Case: Perform PCA on the filtered data. How much variance is explained by the first two principal components?

```
pca <- prcomp(t(assay(rld)[select,]))
summary(pca)
plot(pca)
```

Use Case: Produce a plot of the first two principal components and colour each point according to whether the particular sample is "treated" or "untreated. Does the plot show clear separation of sample groups?

```
pca$x
sampcols <- c(rep("blue",3),rep("red",4))
```

```
plot(pca$x[,1], pca$x[,2],pch=16,col=sampcols)
```

At this point, we introduce the `plotPCA` function which automates the steps we have just performed and produces and attractive plot. The plot is produced by the popular *ggplot2*, and later-on we will see how to customise this plot

Use Case: Use the in-built `plotPCA` function to produce the PCA plot

```
plotPCA(rld, intgroup=c("condition", "type"))
```

1.5 Re-visiting our ESCC dataset

1.5.1 Pre-processing

In the previous DE practical, we have already gone through the processing of this dataset. We can repeat the steps now, if you have closed your RStudio session since the DE practical.

```
library(DESeq2)
load("Day2/Counts.RData")
#Load data
Counts <- tmp$counts
colnames(Counts) <- c("16N", "16T", "18N", "18T", "19N", "19T") #Rename the columns
Coldata <- data.frame(sampleReplicate=c("16", "16", "18", "18", "19", "19"),
sampleType=c("N", "T", "N", "T", "N", "T"))
rownames(Coldata) <- c("16N", "16T", "18N", "18T", "19N", "19T")

deSeqData <- DESeqDataSetFromMatrix(countData=Counts, colData=Coldata,
              design= ~sampleReplicate + sampleType)
deSeqData <- deSeqData[rowSums(counts(deSeqData))>1,]
deSeqData <- estimateSizeFactors(deSeqData)
```

1.5.2 Sample Count Heatmap

Use Case: Produce a heatmap of the normalized counts of the most expressed genes in the ESCC dataset

```
nt <- normTransform(deSeqData) # defaults to log2(x+1)
select <- order(rowMeans(counts(deSeqData,normalized=TRUE)),decreasing=TRUE)[1:20]

log2.norm.counts <- assay(nt)[select,]
df <- as.data.frame(colData(deSeqData)[,c("sampleReplicate","sampleType")])
pheatmap(log2.norm.counts, cluster_rows=FALSE, show_rownames=FALSE,
cluster_cols=FALSE, annotation_col=df)
```

```
pheatmap(log2.norm.counts, cluster_rows=FALSE, show_rownames=FALSE,
cluster_cols=TRUE, annotation_col=df)
```

1.5.3 Sample Distances Heatmap

Use Case: Perform the `rlog` transformation and make a heatmap of the pairwise sample distances.

```
rld <- rlog(deSeqData)
sampleDists <- dist(t(assay(rld)))
sampleDists

sampleDistMatrix <- as.matrix(sampleDists)
rownames(sampleDistMatrix) <- paste(rld$sampleReplicate, rld$sampleType, sep="-")
colnames(sampleDistMatrix) <- NULL
colors <- colorRampPalette( rev(brewer.pal(9, "Blues")) )(255)
pheatmap(sampleDistMatrix,
clustering_distance_rows=sampleDists,
clustering_distance_cols=sampleDists,
col=colors)
```

1.5.4 PCA

Use Case: Perform a PCA analysis and visualise the results

```
plotPCA(rld, intgroup=c("sampleType","sampleReplicate"))
```

The PCA plot is configurable provided we know a tiny bit about *ggplot2*.

Use Case: Re-do the PCA analysis, but this time save the PCA results as an object

```
pcData <- plotPCA(rld, intgroup=c("sampleType","sampleReplicate"),returnData=TRUE)
pcData
```

Use Case: Visualise the PCA result, but colour each point according to sample group and plot a different shape for each patient in the study

```
library(ggplot2)
percentVar <- round(100*attr(pcData, "percentVar"))
ggplot(pcData, aes(x=PC1,y=PC2,color=sampleType,shape=sampleReplicate))+
  geom_point(size=5)+
  xlab(paste0("PC1: ", percentVar[1], "% variance")) +
  ylab(paste0("PC2: ", percentVar[2], "% variance"))
```

1.5.5 Heatmap of DE genes

```
deSeqData <- estimateDispersions(deSeqData)
mcols(deSeqData)
deSeqData <- nbinomWaldTest(deSeqData)
res <- results(deSeqData)
```

```
res.sig <- res[which(res$padj < 0.05),]
N <- 100
res.sig.ord <- res.sig[order(res.sig$padj,decreasing = FALSE),]
topNGenes <- rownames(res.sig.ord)[1:N]
```

```
pheatmap(assay(nt)[match(topNGenes, rownames(assay(nt))),],annotation_col=df)
```

1.5.6 Heatmap of particular gene set

2 GO analysis

```
load("Day2/edgeRAnalysis_ALLGENES.RData")
head(y)
```

```
genes <- as.integer(y$FDR < 0.05)
names(genes) <- rownames(y)
```

```
library(goseq)
head(supportedGenomes())[,1:4]
```

```
head(supportedGeneIDs(),n=12)[,1:4]
```

```
pwf <- nullp(genes ,"hg19","knownGene")
head(pwf)
```

```
go <- goseq(pwf, "hg19","knownGene")
head(go)
```