Annotation and Visualisation of sequencing data in Bioconductor

Mark Dunning

November 3, 2013

Introduction

This practical will use the data from the Pasilla dataset, which you will have already analysed in the RNA-seq pratical. We will assume that the gene counts data have already been analysed. **You do not need to type this code, it is for your reference only.**

```
> library(DESeq)
> datafile = system.file("extdata/pasilla_gene_counts.tsv", package = "pasilla")
> pasillaCountTable = read.table(datafile, header = TRUE, row.names = 1)
> pasillaDesign = data.frame(row.names = colnames(pasillaCountTable),
+     condition = c("untreated", "untreated", "untreated", "untreated",
+         "treated", "treated", "treated"), libType = c("single-end",
+         "single-end", "paired-end", "paired-end", "single-end",
+         "paired-end", "paired-end"))
> pairedSamples = pasillaDesign$libType == "paired-end"
> condition = factor(c("untreated", "untreated", "treated", "treated"))
> countTable = pasillaCountTable[, pairedSamples]
> condition = factor(c("untreated", "untreated", "treated", "treated"))
> cds = newCountDataSet(countTable, condition)
> cds = estimateSizeFactors(cds)
> cds = estimateDispersions(cds)
> res = nbinomTest(cds, "untreated", "treated")
> topHits <- res[order(res$padj, decreasing = F)[1:10], ]
```

The pasillaRes object has been saved in the GeneticsHTSCourse2013 package that accompanies this course.

Use Case: Load the pasilla analysis and familiarise yourself with the contents. Create a data frame of the 100 most significant results.

```
> library(GeneticsHTSCourse2013)
> data(pasillaRes)
> topHits <- pasillaRes[order(pasillaRes$padj, decreasing = F)[1:100],
+     ]
```

Annotation

The biomaRt package provides an interface to the biomart website www.biomart.org. Although the databases stored in biomart, the biomaRt package allows the data to be accessed in a consistent manner and without having write complex SQL queries or understand the underlying schema. The databases available through BioMart can be obtained with the listMarts function

BiomaRt

Use Case: Get a list of all databases available through the biomaRt package and connect to the Drosophilla one.

```
> library(biomaRt)
> listMarts()
> ensembl <- useMart("ensembl", dataset = "dmelanogaster_gene_ensembl")
> attr <- listAttributes(ensembl)
> head(attr, 10)
```

The function getBM is used to build-up queries in biomart. We must specify one or more attributes that we want to retreive. If no values or filters are specified, then all values will be returned.

Use Case: Retrieve the names of all genes for Drosophilla. How many genes are there?

```
> allGenes <- getBM(attributes = c("ensembl_gene_id"), mart = ensembl)
> length(allGenes)
```

When restricting the search to particular query items (e.g. genes) we need to specify both filters and values.

Use Case: Retrieve the ensembl, transcript and peptide ID for the first 10 genes

```
> filt <- listFilters(ensembl)
> head(filt, 10)
> getBM(attributes = attr[1:3, 1], filters = "ensembl_gene_id",
+     values = allGenes[1:10], mart = ensembl)
```

For situations when we want to specify multiple filters, we have to supply the values in list form.

Use Case: Retrieve all the genes between 1100000 and 1250000 on chromsome X.

```
> getBM(attributes = "ensembl_gene_id", filters = c("chromosome_name",
+     "start", "end"), values = list("X", 1100000, 1250000), mart = ensembl)
```

Use Case: Annotate the top hits from the DEseq analysis with their genomic locations and external IDs and create a merged table containing both DESeq results and the annotations.

```
> myInfo <- getBM(attributes = c("flybase_gene_id", "chromosome_name",
+     "band", "start_position", "end_position", "external_gene_id"),
+     filters = "flybase_gene_id", values = topHits[, 1], mart = ensembl)
> myInfo
> annotatedHits <- merge(topHits, myInfo, by.x = 1, by.y = 1)
> head(annotatedHits)
```

Genome Representation

Use Case: Load the package that provides the genome representation for Drosophila

```
> library(BSgenome)
> available.genomes()
> library(BSgenome.Dmelanogaster.UCSC.dm3)
> Dmelanogaster
```

Use Case: Retrieve the sequence for chromosome X and subset to get the sequence for gene FBgn0040357.

```
> geneInfo <- getBM(attributes = c("chromosome_name", "start_position",
+     "end_position"), filters = "ensembl_gene_id", values = "FBgn0040357",
+     ensembl)
> gr <- GRanges("chrX", IRanges(geneInfo$start_position, geneInfo$end_position))
> seq <- getSeq(Dmelanogaster, gr)
> seq
```

Use Case: Retrieve sequences for the annotated results from the DEseq analysis and write them out as a Fasta file

```
> myseqs <- getSeq(Dmelanogaster, paste("chr", annotatedHits$chromosome_name,
+     sep = ""), annotatedHits$start_position, annotatedHits$end_position)
> myseqs
> names(myseqs) <- annotatedHits[, 1]
> writeXStringSet(myseqs, file = "topGenes.fa")
```

Various Biostrings operations could now be performed on these sequences. Some suggestions are given below.

```
> substr(myseqs, 1, 10)
> translate(myseqs)
> gcFunction(myseqs)
```

1 Alternative Feature Counting

In the analysis of the pasilla dataset, we allocated reads to genes. However, in this section we will explore the ways of counting reads that align to other genomic regions of interest. In this case we will use exons. An example bam file is provided in the pasillaBamSubset package, but only reads on chromosome 4 are provided.

1.1 Transcript Annotation

In order to use the example bam file, which is chromosome 4 only, we will have to get the IDs of genes that occur on chromosome 4. We illustrate this using the org.Dm.eg.db package.

Use Case: Use the appropriate organism package to retrieve the ensembl IDs of all genes located on chromosome 4 of Drosophilla.

```
> library(org.Dm.eg.db)
> dm <- org.Dm.eg.db
> cols(dm)
> keytypes(dm)
> chr4Genes <- select(dm, cols = "ENSEMBL", keytype = "CHR", keys = "4")
> head(chr4Genes)
> chr4ID <- chr4Genes[, 2]
```

Bioconductor provide a number of pre-built packages that have a database of all transcript information for a particular organism. The package we are going to use is

TxDb.Dmelanogaster.UCSC.dm3.ensGene.

As its name suggests, this package was built from the UCSC tables for the dm3 genome and uses Ensembl genes as an identifier. Convenient functions exist that can return the gene structure as a GRanges object.

Use Case: Get all the exons for all genes on chromosome 4

```
> library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)
> tx <- TxDb.Dmelanogaster.UCSC.dm3.ensGene
> exo <- exonsBy(tx, "gene")
> exo
> length(exo)
> exo4 <- exo[names(exo) %in% chr4ID]
> length(exo4)
```

1.2 Exon Counting

Use Case: Read the example bam file from the pasillaBamSubset package. Verify that only chromosome 4 reads are included in the object

```
> library(pasillaBamSubset)
> bam <- readGappedAlignments(untreated3_chr4(), use.name = TRUE)
> table(seqnames(bam))
```

We are now in a position to begin overlapping reads with exons. You should have already seen the findOverlaps function that will report the amount of ovelap between two sets of ranges

Use Case: Use the findOverlaps function to count the number of reads covering each exon of FBgn0002521.

```
> olaps <- findOverlaps(exo4[["FBgn0002521"]], bam)
> olaps
> bam[subjectHits(olaps)]
```

summarizeOverlaps extends findOverlaps by providing options to resolve reads that overlap multiple features. Each read is counted a maximum of once. Different modes of counting are available. See the help page for summarizeOverlaps for more details.

Use Case: Repeat the counting for all genes on chromosome 4

```
> olapGenes <- summarizeOverlaps(exo4, bam)
> countGenes <- assays(olapGenes)$counts
> countGenes
```

The count vector has entries for each gene on chromosome 4. If we wanted counts for all exons, then we would have to supply a slightly different argument to the function with

Use Case: Obtain per-exon counts for each gene

```
> exonRanges <- unlist(exo4)
> olapExon <- summarizeOverlaps(exonRanges, bam)
> countExons <- assays(olapExon)$counts
> countExons
```

The summarizeOverlaps function can also take the location of one or more bam files as an argument. This means that the processing of reads and counting is handled in one step.

Use Case: Obtain counts of chromosome 4 genes in the bam files included with the pasillaSubset package.

```
> fls <- c(untreated3_chr4(), untreated1_chr4())
> names(fls) <- basename(fls)
> multiOlap <- summarizeOverlaps(exo4, BamFileList(fls))
> head(assays(multiOlap)$counts)
```

1.3 Importing Tracks

Genome browser or other tracks are often distributed as files in the formats gff, bed, wig etc. The package rtracklayer provides infrastrucutre to import files in these formats and create a GRanges representation.

Use Case: Read the gff file supplied with the pasilla package and use these intervals to produce a table of overlaps.

```
> library(rtracklayer)
> gffFile <- "Dmel.BDGP5.25.62.DEXSeq.chr.gff"
> gff <- paste(system.file("extdata", package = "pasilla"), gffFile,
+     sep = "/")
```

```
> read.table(gff, nrows = 10, sep = "\t")
> gffRange <- import(gff)
> gffRange
> gffOverlap <- summarizeOverlaps(gffRange, bam)
```

Use Case: Find out what tracks are available for the dm3 genome. Download the RepeatMasker track.

```
> mySession <- browserSession()
> track.names <- trackNames(ucscTableQuery(mySession))
> repeats <- getTable(ucscTableQuery(mySession, track = "rmsk",
+     table = "rmsk"))
> head(repeats)
```

1.4 Exporting tracks

It is also possible to save the results of a Bioconductor analysis in a browser to enable interactive analysis and integration with other data types, or sharing with collaborators. We shall use the bed format for illustration.

Use Case: Select all significant genes from the pasilla analysis and annotate their genome location. Create a GRanges representation of the locations.

```
> sigResults <- pasillaRes[which(pasillaRes$padj < 0.1), ]
> myInfo <- getBM(attributes = c("flybase_gene_id", "chromosome_name",
+     "band", "start_position", "end_position", "external_gene_id"),
+     filters = "flybase_gene_id", values = sigResults[, 1], mart = ensembl)
> myInfo <- merge(sigResults, myInfo, by.x = 1, by.y = 1)
> sigRanges <- GRanges(paste("chr", myInfo$chromosome_name, sep = ""),
+     IRanges(start = myInfo$start_position, end = myInfo$end_position,
+         names = myInfo[, 1]), padj = myInfo$padj, logfc = myInfo$log2FoldChange)
```

Rather than just representing the genomic locations, the .bed format is also able to colour each range according to some property of the analysis (e.g. direction and magnitude of change) to help highlight particular regions of interest. A score can also be displayed when a particular region is clicked-on.

Use Case: Create a score from the p-values and colour scheme based on the fold-change.

```
> Score <- log10(values(sigRanges)$padj)
> rbPal <- colorRampPalette(c("red", "blue"))
> logfc <- pmax(values(sigRanges)$logfc, -3)
> logfc <- pmin(logfc, 3)
> Col <- rbPal(10)[as.numeric(cut(logfc, breaks = 10))]
```

The colours and score can be saved in the GRanges object and score and itemRgb columns respectively and will be used to construct the track when the export is called.

Use Case: Export the signifcant results from the DE analysis as a .bed track.

```
> values(sigRanges)$score <- Score
> values(sigRanges)$itemRgb <- Col
> export(sigRanges, con = "topHits.bed")
```

The export of other formats such as *gff*, *bedGraph*, *wig* and *bigwig* is also possible with rtracklayer. We can also call the UCSC browser directly to display the tracks. See the rtracklayer vignette for details.

1.5 ggbio

We will now take a brief look at one of the visualisation packages in Bioconductor that takes advantages of the GenomicRanges and GenomicFeatures object-types. The documentation for ggbio is very extensive and contains lots of examples.

`http://www.tengfei.name/ggbio/docs/`
Extra background reading can be found on the ggplot2 website; the package on which the principles of ggbio is based.
`http://ggplot2.org/`
The Gviz package another Bioconductor package that specialising in genomic visualisations, but we will not explore this package in the course. A useful function within ggbio is autoplot, which will construct an appropriate plot based on the object-type of the input. For example, if we pass a GRanges object to the function it will plot the genomic locations on a linear scale and label chromosome and positional information on the plot. The style of the plot can be changed by the layout argument.

Use Case: Plot the locations of the significant hits from the DE analysis. Try the default, karyogram and circle layouts

```
> library(ggbio)
> autoplot(sigRanges)
> autoplot(sigRanges[seqnames(sigRanges) == "chrX"])
> autoplot(sigRanges, layout = "karyogram")
> autoplot(sigRanges, layout = "circle")
```

The Manhattan plot 1is a common way of visualising genome-wide results, and this is implemented as the `plotGrandLinear` function. We have to supply a value to display on the y-axis. This is done by using the `aes` function, which is inherited from ggplot2.

Use Case: Plot the genomic locations of the significant genes on a linear scale. Modify the colour scheme to show whether each gene is up- or down-regulated

```
> plotGrandLinear(sigRanges, aes(y = score))
> values(sigRanges)$Up <- logfc > 0
> plotGrandLinear(sigRanges, aes(y = score, col = Up))
```

ggbio is also able to plot the structure of genes according to a particular model represented by a `GenomicFeatures` object.

Use Case: Find the gene which has the most exon counts in the pasilla bam. Plot the struture of this gene.

```
> myGene <- names(exonRanges)[which.max(countExons)]
> autoplot(tx, which = exo4[[myGene]])
```

Use Case: Plot the number of reads in this region surrounding the gene.

```
> myreg <- flank(reduce(exo4[[myGene]]), 500, both = T)
> bamSubset <- bam[bam %over% myreg]
> autoplot(bamSubset)
```

Use Case: Repeat the plot, but with a smoothed coverage

```
> autoplot(bamSubset, stat = "coverage")
```

The plots produced by ggbio (and indeed ggplots2 on which the package is based) are not in standard R graphics format, so techniques like par(mfrow) for arranging plots on the same page are not applicable. However, ggbio allows the plots to be saved as objects, which can later be arranged by specialist functions such as tracks. This function automatically aligns the x-axis of each plot to be on the same scale.

Use Case: Make a combined plot of the sample information and transcripts

```
> p1 <- autoplot(tx, which = myreg)
> p2 <- autoplot(bamSubset, stat = "coverage")
> tracks(p1, p2)
```

The combined plot can also be embellished by adding an Ideogram to show the location of the region being plotted in relation to the chromosome.

Use Case: Get the ideogram data for the drosophilla genome and add to the plot

```
> plotIdeogram(genome = "dm3")
> plotIdeogram(genome = "dm3", subchr = "chr4")
> start <- min(start(bamSubset))
> end <- max(end(bamSubset))
> id <- plotIdeogram(genome = "dm3", subchr = "chr4", zoom.region = c(start,
+     end))
> tracks(id, p1, p2)
```