Annotation and down-stream analysis – lab

Martin Morgan*

June 20-23, 2011

Annotations are used to understand results of an analysis, e.g., relating microarray probe sets to the gene names or symbols that they interrogate, or placing genes into biochemical pathways or gene ontologies. Sequence data differs from microarrays in that regions emerging from the analysis are not strictly pre-determined (as, e.g., by probes placed on a microarray); annotations are used more directly in analysis (e.g., defining regions in which 'counts' are determined) or as a way to place new information into a spatial context (e.g., proximity of ChIP peaks to existing genes; assessing the plausibility of novel transcripts).

This lab explores how to effectively use *Bioconductor* annotation resources.

```
> library(EMBL2011)
```

1 *Bioconductor* resources

Bioconductor provides annotation resources for organisms (the '*org.**' packages), homology, microarrays, gene ontology (*GO.db*) and KEGG pathways (*KEGG.db*), and many others. The *BSgenome* package provides whole-genome sequences for model and other organisms. The *GenomicFeatures* package can be used to retrieve and curate structural information from the UCSC, Biomart and other web sites.

1.1 Genes

Exercise 1

Basic annotations about genes are found in a suite of 'org' packages, e.g., org.Dm.eg.db.

a. *Load the org.Dm.eg.db package, list the symbols defined in this package (using the appropriate name returned by* `search()` *as the first argument to* `ls`*).*

b. *Use the function* `org.Dm.eg()` *and the R help system (e.g.,* `?org.Dm.egPATH` *to discover curation (metadata) details about the data in this package.*

*mtmorgan@fhcrc.org

Solution: Load the package as:

```
> library(org.Dm.eg.db)
```

Metadata is available using `org.Dm.eg()`. Maps and other information provided by the package are on the search path. Details about individual maps can be found on the corresponding help page.

```
> org.Dm.eg()                  # metadata
> search()                     # packages on the 'search' path
> ls("package:org.Dm.eg.db")   # available maps
> ?org.Dm.egPATH
```

Exercise 2
org. packages contain a number of 'bimaps'. Each bimap describes the relationship between an 'Lkey' and an 'Rkey' (roughly, key and corresponding value in a hash table). The central key used in a package is indicated in the package name; *.*.eg.db packages use the* entrez gene identifier.

a. *What is the bimap relating Entrez and Ensembl gene identifiers?*

b. *The Lkey can be used to subset a bimap using the* `[`*operator. Create a sub-map of the Entrez=to-Ensembl map for the Entrez identifiers* `c("32007", "32008", "38248", "39611")`.

c. *Query the sub-map for the 'Lkeys' present using* `mappedLkeys`*, and the 'Rkeys' with* `mappedRkeys`.

d. *Many maps can be reversed, e.g., to map from Ensembl to Entrez identifiers. Use* `revmap` *to reverse the Entrez-to-Ensembl map, and query the reversed map with known Ensembl identifiers (e.g., the ids from the previous question). The 'L' and 'R' in* `mappedLkeys` *and* `mappedRkeys` *functions refer to the original map.*

e. *Maps can be expanded to reveal their detail with, e.g.,* `toTable` *or* `as.list`*. Display the data frame (*`toTable`*) or list (*`as.list`*) associated with the Ensembl-to-Entrez map subset for a few known Ensembl identifiers.*

f. *Individual entries in a bimap can be extracted with* `[[`*use this to extract the gene name associated with Entrez identifier* `32008`.

Solution: The bi-map between Entrez- and Ensembl gene identifiers is `org.Dm.egENSEMBL`. Subset the map with the usual single square bracket `[`with arguments being Entrez gene identifiers. Retrieve the 'Lkeys' in the map (should be the same as `egids`), and the 'Rkeys' to which they map.

```
> org.Dm.egENSEMBL
> egid <- c("32007", "32008", "38248", "39611")
> (eg2ens <- org.Dm.egENSEMBL[egid])
> mappedLkeys(eg2ens)
> (ensid <- mappedRkeys(eg2ens))
```

Reverse the map and go from Ensembl ids to Entrez ids. The 'Lkeys' and 'Rkeys' refer to 'left' and 'right' in the original (non-reversed) map.

```
> (ens2eg <- revmap(org.Dm.egENSEMBL)[ensid])
> mappedLkeys(ens2eg)
```

A fuller representation of the map can be obtained.

```
> toTable(ens2eg)
> str(as.list(ens2eg))
```

Double square brackets extract a single entry.

```
> org.Dm.egGENENAME[["32008"]]
```

Exercise 3
Four genes of interest identified in Figure 3 of [1] are `Ant2`, `sesB`, `bmm`, *and* `dre4`. *These identifiers are gene 'symbols', an easily remembered but notoriously poor (because there is a many-to-many map between symbol and gene) choice. Use the* `org.Dm.egALIAS2EG` *map (the Lkeys are still Entrez identifiers) to translate symbol names to Entrez identifiers, and Entrez identifiers to Ensembl identifiers.*

Solution:

```
> symid <- c("Ant2", "sesB", "bmm", "dre4")
> (sym2eg <- org.Dm.egALIAS2EG[symid])
> toTable(sym2eg)
> egid <- mappedLkeys(sym2eg)
> eg2ens <- org.Dm.egENSEMBL[egid]
> (ensid <- mappedRkeys(eg2ens))
> names(ensid) <- symid
> ensid
```

Exercise 4
Use `toTable` *on each translation to display the map as a data frame, and then* `merge` *both data frames to create a summary of all three identifiers.*

Solution:

```
> (tbl <- merge(toTable(sym2eg), toTable(eg2ens)))
```

1.2 Transcripts

Exercise 5

a. *(This part requires internet access) Use* `makeTranscriptDbFromUCSC` *to retrieve information about the physical structure of genes from the UCSC genome browser. Retrieve information for the* `dm3` *genome (*D. melanogaster*, version 3) and the* `ensGene` *table.*

b. *Save the data as a SQLite data base to a local disk. Load it again. Query the object for information about its provenance, using* `metadata`*.*

c. *As an advanced exercise, query the data base from the command line, using SQLite (if installed).*

Solution: The following command retrieves and save the current information (`tempdir()` is a temporary location that will be destroyed when the *R* session ends). Be sure to name the file with the `.sqlite` extension.

```
> (txdb <- makeTranscriptDbFromUCSC("dm3", "ensGene"))
> (txdbFile <- file.path(tempdir(), "dm3.ensGene.txdb.sqlite"))
> saveFeatures(txdb, txdbFile)
```

The *EMBL2011* package contains a snapshot of this data base; load that into the current *R* session.

```
> txdbFile <- system.file("extdata", "dm3.ensGene.txdb.sqlite",
+                         package="EMBL2011")
> txdb <- loadFeatures(txdbFile)
> head(metadata(txdb))
```

Exercise 6

a. *Query the* `txdb` *object for a list of transcripts grouped by gene using the* `transcriptsBy`*.*

b. *Try accessing the elements of the returned list, e.g., selecting the first or first several transcripts, or the transcripts associated with the* `dre4` *symbol from a previous exercise.*

c. *Extract the transcript names from the object, both for a single transcript and for all transcripts. To extract one transcript, use the* `[[]]` *notation.*

d. *Query the* `txdb` *object for the sequences that are currently 'active' using* `isActiveSeq`*. Change the active sequences to exclude the 'Het' and 'extra' chromosomes, and retrieve the list of transcripts grouped by gene, and exons grouped by gene.*

Solution: Extracted transcripts are returned as a *GRangesList*. The names of the list elements are the Ensembl (in this case) gene names. The transcript

name and (internal) identifier are also returned. The metadata from the data base is present in the *GRangesList*.

GRangesList are like a regular list, in terms of sub-setting, e.g., by name, numeric, or logical index. Each element of the *GRangesList* is a *GRanges* instance.

```
> tx <- transcriptsBy(txdb, "gene")
> length(tx)
> head(tx, 2)
> head(metadata(tx)[[1]])
```

Transcript names are properties of the elements of the *GRangesList*. To extract the transcript names of a single element, extract one element, then query the *GRanges* for its `values()` and subset accordingly.

```
> ## one transcript
> tx["FBgn0002183"]
> values(tx[["FBgn0002183"]])$tx_name
```

The commands to extract *all* transcript names requires either that each element of the *GRangesList* be extracted and queried or that the *GRangesList* be unlisted (to a *GRanges* instance) and then queried.

```
> ## slow: sapply(tx, function(elt) values(elt)$tx_name)
> txx <- unlist(tx, use.names=FALSE)  # GRanges
> txnm <- values(txx)$tx_name
> length(txnm)
> head(txnm, 4)
> ## advanced: grouped by gene
> txnmByGene <- split(txnm, rep(names(tx), elementLengths(tx)))
> txnmByGene[1:3]
```

Sequences can be 'active' or not; only transcripts or exons from active sequences are extracted from the data base.

```
> isActiveSeq(txdb)
> nms <- grep("(Het|extra)", names(isActiveSeq(txdb)), value=TRUE)
> isActiveSeq(txdb)[nms] <- FALSE
> isActiveSeq(txdb)
> tx <- transcriptsBy(txdb, "gene")
> ex <- transcriptsBy(txdb, "gene")
```

Exercise 7

This exercise illustrates how annotation information can be integrated with the Rsamtools package. The goal is to use transcript coordinates as a basis for querying a BAM file. Specifically, we ask about the insert width of paired end reads mapping to genes consisting of a single exon and transcript. The naive motivation is that these represent an appropriate null against which to compare insert widths observed in genes with more complicated structure.

a. *Use* `transcriptsBy` *to extract a GRangesList describing the transcripts present in each gene. Use* `elementLengths` *on this object to identify genes that consist of exactly one transcript.*

b. *Similarly, use* `exonsBy` *to extract a GRangesList describe the exons present in each gene. Use* `elementLengths` *on this object to identify genes that consist of exactly one exon.*

c. *Use the R function* `intersect` *to identify the genes that have exactly one transcript, encoded by exactly one exon.*

d. *Use the ranges of the single-exon, single-transcript genes to query BAM files for insert width. This is tricky.*

   (a) *Recall from previous exercises how to list and open BAM files for our lab.*

   (b) *Remove the lanes that are not paired-end; we cannot determine insert width from this data.*

   (c) *Create a ScanBamParam object with a* `which` *argument that restricts the query to the regions corresponding to the single-exon, single-transcript genes.*

   (d) *Apply* `insertSize` *to each BAM file.*

e. *As a final component of this exercise, create an lattice* `densityplot` *that describes insert width.*

Solution: Identify the names of the transcripts of genes with exactly one transcript, and the genes with single exons...

```
> ## transcripts
> tx <- transcriptsBy(txdb, "gene")
> tx1idx <- elementLengths(tx) == 1
> tx1nm <- names(tx)[tx1idx]
> ## exons
> ex <- exonsBy(txdb, "gene")
> ex1idx <- elementLengths(ex) == 1
> ex1nm <- names(ex)[ex1idx]
```

Find the intersection, i.e., genes with a single transcript and a single exon.

```
> gn1 <- intersect(tx1nm, ex1nm)
> length(gn1)
```

Use the ranges of these genes to query the bam files for insert widths. Start by listing and opening the (paired-end) BAM files.

```
> if (interactive())
+     bamFiles <- file.choose()
> fls <- list.files(bamFiles, "bam$", full=TRUE)
```

```
> bam <- open(BamFileList(fls))
> names(bam) <- sub(".bam", "", basename(fls))
> bam2 <- bam[-c(1, 4:5)]                 # drop single-end files
> snms <- seqnames(seqinfo(bam[[1]]))
```

Create a *ScanBamParam* instance where `which` corresponds to the single-exon, single-transcript genes.

```
> which0 <- unlist(ex[gn1], use.names=FALSE)
> which <- keepSeqlevels(which0, snms) # drop sequences not in bam files
> param <- ScanBamParam(which=which)
> ## hist(log10(width(which)))
```

Calculate insert size using `insertSize` (developed in a previous lab) on each bam file. The code below uses `mclapply`, which is like `lapply` but runs across multiple cores (`mclapply` is provided by the *multicore* package; this package is not, unfortunately, available on Windows).

```
> doit <-
+     if (suppressWarnings(require("multicore"))) {
+         mclapply
+     } else lapply
> iwd0 <- doit(as.list(bam2), insertSize, param=param)
> iwd <- lapply(iwd0, unlist)
```

Finally, display the distribution of insert sizes by converting the data to a 'long' data frame and displaying using the `densityplot` function from the *lattice* package.

```
> df <- do.call(make.groups, iwd)
> densityplot(~data, group=which, df[df$data < 600,],
+             plot.points=FALSE,
+             auto.key=list(lines=TRUE, points=FALSE, columns=2),
+             file="single-exon-inserts.pdf")
```

The result of this operation is presented in Figure 1.

2 Internet resources with *Bioconductor*

2.1 Biomart

Rich annotation resources are available on the web. Many resources are accessible in *Bioconductor* through special purpose packages, e.g., *biomaRt* for accessing the Biomart collection of resources, or *GEOquery* and *ArrayExpress* for accessing the corresponding microarray repositories. Flexible programming tools such as the *XML* and *RCurl* packages provide programmatic access to virtually any internet resource. Secondary internet resources are extensive and

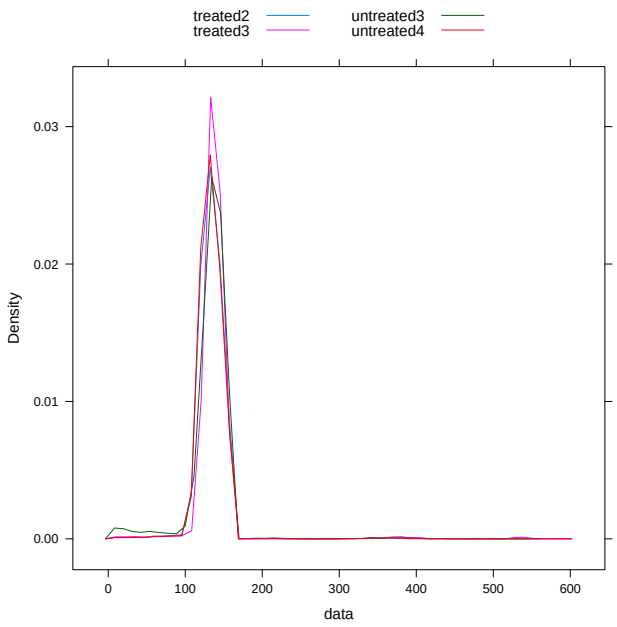

Figure 1: Insert widths in single-exon, single-transcript genes.

'current', but are often transient or surprisingly intermittent in their availability, and frequently lack version information required for reproducible research. This can make internet resources frustrating to work with during analysis.

This section of the lab uses the *biomaRt* package to retrieve information about DNA sequences of particular genic locations, and compares the results with those accessible using *Bioconductor* annotation resources.

Exercise 8

The biomaRt package provides a very nice way to query 'marts' of data sets that conform to particular standards.

a. *'Discover' the marts (collections of data sets) available at the default Biomart location using the biomaRt function* `listMarts`*. Select the* `ensembl` *mart with the* `useMart` *function.*

b. *Each mart has several data sets associated with it. List available data sets in the* `ensembl` *mart using the* `listDatasets` *function. Select the* `"dmelanogaster_gene_ensembl"` *data set using the* `useDataset` *function.*

c. *Data sets have* attributes *that one can retrieve, and* filters *that restrict the records that are returned. List the attributes and filters associated with the data set you are using with the* `listAttributes` *and* `listFilters` *functions.*

Solution: Load the *biomaRt* package and discover available marts; select the `ensembl` mart.

```
> library(biomaRt)
> head(listMarts(), 4)
> mart <- useMart("ensembl")
```

Select the data set with *Drosophila* Ensembl genes.

```
> dsets <- listDatasets(mart)
> head(dsets, 4)
> dset <- useDataset("dmelanogaster_gene_ensembl", mart)
```

Query the data set for the attributes and filters it supports.

```
> head(listAttributes(dset))
> head(listFilters(dset), 10)
```

Exercise 9

This exercise uses the biomaRt package to retrieve GC content of the four genes (`<<symid>>`) from Figure 3 of [1].

Begin by retrieving Biomart information about GC content of the genes we are interested in.

a. *Select attributes that return the Ensembl gene and transcript identifiers and the Entrez gene identifier.*

b. *Use `entrezgene` as the `filter` argument, and the Entrez gene identifiers `<<egid>>` as the `values` argument.*

c. *Retrieve the relevant annotation using `getBM`.*

Next...

a. *Refine the query to return Ensembl gene identifiers, chromosome name, start and end position, and strand information. Importantly, also return GC content.*

b. *Represent the results as a `GRanges` instance.*

Finally...

a. *Load the BSgenome package containing the* D. melanogaster *genome, BSgenome.Dmelanogaster.UCSC.dm3.*

b. *Use the `getSeq` function to retrieve sequences corresponding to the biomaRt query.*

c. *Use `alphabetFrequency` along with `rowSums` to calculate the GC content in the sequences returned by `getSeq`. How does this compare with the GC content retrieved using biomaRt?*

Solution: Set the attributes and filters, and retrieve the annotations.

```
> attrs <- c("ensembl_gene_id", "ensembl_transcript_id", "entrezgene")
> filters <- "entrezgene"
> values <- c("32007", "32008", "38248", "39611")
> (anno <- getBM(attrs, filters, values, mart=dset))
> ## compare with unlist(tx[ensid])
```

Refine the query to return genes, their genomic location, and GC content.

```
> attrs <- c("ensembl_gene_id", "chromosome_name",
+            "start_position", "end_position",
+            "strand", "percentage_gc_content")
> (anno <- getBM(attrs, filters, values, mart=dset))
```

Represent the results as a *GRanges* instance.

```
> (gr <- with(anno,
+             GRanges(sprintf("chr%s", chromosome_name),
+                     IRanges(start_position, end_position,
+                             names=ensembl_gene_id),
+                     strand=ifelse(strand==-1, "-", "+"))))
```

Extract the relevant sequence from the relevant *BSgenome* package, and calculate GC content.

```
> library(BSgenome.Dmelanogaster.UCSC.dm3)
> (seq <- getSeq(Dmelanogaster, gr))
> alf <- alphabetFrequency(seq, baseOnly=TRUE)
> rowSums(alf[,c("G", "C")]) / rowSums(alf)
```

As an advanced exercise: from the BAM files, what is the GC content of reads aligning to these regions?

References

[1] A. N. Brooks, L. Yang, M. O. Duff, K. D. Hansen, J. W. Park, S. Dudoit, S. E. Brenner, and B. R. Graveley. Conservation of an RNA regulatory map between Drosophila and mammals. *Genome Res.*, 21:193–202, Feb 2011.

A Appendix