The **rtracklayer** package

Michael Lawrence

October 28, 2009

1 Introduction

The **rtracklayer** package is an interface (or *layer*) between **R** and genome browsers. Its main purpose is the visualization of genomic annotation *tracks*, whether generated through experimental data analysis performed in R or loaded from an external data source. The features of **rtracklayer** may be divided into two categories: 1) the import/export of track data and 2) the control and querying of external genome browser sessions and views.

The base track data structure is *RangedData*, as defined by the **IRanges** package. **rtracklayer** supports the import and export of tracks from and to files in various formats, see Section 2.1.4. All positions in a *RangedData* should be 1-based, as in R itself.

The **rtracklayer** package currently interfaces with the **UCSC** web-based genome browser. Other packages may provide drivers for other genome browsers through a plugin system. With **rtracklayer**, the user may start a genome browser session, create and manipulate genomic views, and import/export tracks and sequences to and from a browser. Please note that not all features are necessarily supported by every browser interface.

The rest of this vignette will consist of a number of case studies. First, we consider an experiment investigating microRNA regulation of gene expression, where the microRNA target sites are the primary genomic features of interest.

2 Gene expression and microRNA target sites

This section will demonstrate the features of **rtracklayer** on a microarray dataset from a larger experiment investigating the regulation of human stem cell differentiation by microRNAs. The transcriptome of the cells was measured before and after differentiation by HG-U133plus2 Affymetrix GeneChip arrays. We begin our demonstration by constructing an annotation dataset from the experimental data, and then illustrate the use of the genome browser interface to display interesting genomic regions in the UCSC browser.

2.1 Creating a target site track

For the analysis of the stem cell microarray data, we are interested in the genomic regions corresponding to differentially expressed genes that are known to be targeted by a microRNA. We will represent this information as an annotation track, so that we may view it in the UCSC genome browser.

2.1.1 Constructing the *RangedData*

In preparation for creating the microRNA target track, we first used **limma** to detect the differentially expressed genes in the microarray experiment. The locations of the microRNA target sites were obtained from MiRBase. The code below stores information about the target sites on differentially expressed genes in the *data.frame* called `targets`, which can also be obtained by entering `data(targets)` when **rtracklayer** is loaded.

```
> library("humanStemCell")
> data(fhesc)
> library("genefilter")
> filtFhesc <- nsFilter(fhesc)[[1]]
> library("limma")
> design <- model.matrix(~filtFhesc$Diff)
> hesclim <- lmFit(filtFhesc, design)
> hesceb <- eBayes(hesclim)
> tab <- topTable(hesceb, coef = 2, adjust.method = "BH",
+     n = 7676)
> tab2 <- tab[(tab$logFC > 1) & (tab$adj.P.Val < 0.01),
+     ]
> affyIDs <- tab2$ID
> library("microRNA")
> data(hsTargets)
> library("hgu133plus2.db")
> entrezIDs <- mappedRkeys(hgu133plus2ENTREZID[affyIDs])
> library("org.Hs.eg.db")
> mappedEntrezIDs <- entrezIDs[entrezIDs %in% mappedkeys(org.Hs.egENSEMBLTRANS)]
> ensemblIDs <- mappedRkeys(org.Hs.egENSEMBLTRANS[mappedEntrezIDs])
> targetMatches <- match(ensemblIDs, hsTargets$target,
+     0)
> targets <- hsTargets[targetMatches, ]
```

The following code creates the track from the `targets` dataset. First, we construct an *IRanges* instance for holding the start, end and name of each target site. This is then passed to the `GenomicData` function, which constructs a *RangedData* instance. The stand information, the name of the microRNA and the Ensembl ID of the targeted transcript are stored as columns in the *RangedData*. The chromosome for each site is passed as the `chrom` argument.

```
> head(targets)
```

```
                name          target chrom     start       end
555774     hsa-miR-16 ENST00000000412    12   8985197   8985217
415091 hsa-miR-509-3p ENST00000003084     7 117095440 117095461
594550    hsa-miR-612 ENST00000003834    17  23750064  23750088
398678 hsa-miR-423-3p ENST00000006015     7  27187935  27187957
607152   hsa-miR-125b ENST00000006101    17  43458623  43458643
608640 hsa-miR-324-3p ENST00000006658    17  45988032  45988051
       strand
555774      -
415091      +
594550      +
398678      -
607152      -
608640      +
```

```
> library(IRanges)
> targetRanges <- IRanges(targets$start, targets$end)
> library(rtracklayer)
> targetTrack <- GenomicData(targetRanges, targets[, c("strand",
+     "name", "target")], chrom = paste("chr", targets$chrom,
+     sep = ""), genome = "hg18")
```

2.1.2 Accessing track information

The track information is now stored in the R session as a *RangedData* object. It holds the feature start and end positions along with any number of data columns. The ranges and data are grouped by chromosome (or other sequence).

The primary feature attributes are the `start`, `end`, `chrom` and `strand`. There are accessors for each of these, named accordingly. For example, the following code retrieves the chromosome names and then start positions for each feature in the track.

```
> head(chrom(targetTrack))
```

```
[1] chr1 chr1 chr1 chr1 chr1 chr1
23 Levels: chr1 chr10 chr11 chr12 chr13 chr14 chr15 chr16 ... chrX
```

```
> head(start(targetTrack))
```

```
[1]  7762840 11957570 91921292 86981576 11009260 54270236
```

The `chrom` and `strand` accessors are defined by the **rtracklayer** package, for use with genomic ranges. They are simple wrappers around `names(targetTrack)` and `targetTrack$strand`, respectively. The strand is just another variable in the data and should be a factor with levels *+*, - and * (either).

Exercises

1. Get the strand of each feature in the track
2. Calculate the length of each feature
3. Reconstruct (partially) the `targets` *data.frame*

2.1.3 Subsetting a *RangedData*

It is often helpful to extract subsets from *RangedData* instances, especially when uploading to a genome browser. The data can be subset though a matrix-style syntax by feature and column. The conventional `[` method is employed for subsetting, where the first parameter, i, indexes the features and j indexes the data columns. Both i and j may contain numeric, logical and character indices, which behave as expected.

```
> first10 <- targetTrack[1:10, ]
> posTargets <- targetTrack[strand(targetTrack) == "+",
+     ]
```

The features may also be indexed by a *IRanges* instance (explained later) or by chromosome. We give an example of the latter below for obtaining all target sites on chromosome 1. The name or index of a chromosome is passed as the sole argument to the `[` function, as in list subsetting.

```
> chr1Targets <- targetTrack["chr1"]
```

Exercises

1. Subset the track for all features on the negative strand of chromosome 2.

2.1.4 Exporting and importing tracks

Import and export of *RangedData* instances is supported in the following formats: Browser Extended Display (BED), versions 1, 2 and 3 of the General Feature Format (GFF), and Wiggle (WIG). Support for additional formats may be provided by other packages through a plugin system.

To save the microRNA target track created above in a format understood by other tools, we could export it as BED. This is done with the `export` function, which accepts a filename or any R connection object as its target. If a target is not given, the serialized string is returned. The desired format is derived, by default, from the extension of the filename. Use the `format` parameter to explicitly specify a format.

```
> export(targetTrack, "targets.bed")
```

To read the data back in a future session, we could use the `import` function. The source of the data may be given as a connection, a filename or a character vector containing the data. Like the `export` function, the format is determined from the filename, by default.

```
> restoredTrack <- import("targets.bed")
```

Exercises

1. Output the track to a file in the "gff" format.
2. Read the track back into R.
3. Export the track as a character vector.

2.2 Viewing the targets in a genome browser

For the next step in our example, we will load the track into a genome browser for visualization with other genomic annotations. The **rtracklayer** package is capable of interfacing with any genome browser for which a driver exists. In this case, we will interact with the web-based **UCSC** browser, but the same code should work for any browser.

2.2.1 Starting a session

The first step towards interfacing with a browser is to start a browser session, represented in R as a *BrowserSession* object. A *BrowserSession* is primarily a container of tracks and genomic views. The following code creates a *BrowserSession* for the **UCSC** browser:

```
> session <- browserSession("UCSC")
```

Note that the name of any other supported browser could have been given here instead of "UCSC". To see the names of supported browsers, enter:

```
> genomeBrowsers()
```

```
[1] "UCSC"
```

2.2.2 Laying the track

Before a track can be viewed on the genome, it must be loaded into the session using the `track<-` function, as demonstrated below:

```
> track(session, "targets") <- targetTrack
```

The *name* argument should be a character vector that will help identify the track within `session`. Note that the invocation of `track<-` above does not specify an upload format. Thus, the default, "auto", is used. Since the track does not contain any data values, the track is uploaded as BED. To make this explicit, we could pass "bed" as the *format* parameter.

Exercises

1. Lay a track with the first 100 features of `targetTrack`

   Here we use the short-cut `[[` syntax for storing the track.

2.2.3 Viewing the track

For **UCSC**, a view roughly corresponds to one tab or window in the web browser. The default view of the track attempts to show the entire track. The view region is determined by a call to `range` on the track object.

```
> range(chr1Targets)

CompressedIRangesList of length 1
$chr1
IRanges of length 1
     start       end     width
[1] 939753 245386490 244446738
```

The returned value from `range` is an instance of the *RangesList* class, which specifies a segment of a genome by its genome ID, chromosome, and start and end positions.

Exercises

1. Get the *RangesList* describing the first feature in `targetTrack`.
2. Get the chromosome ID of that segment.

The target sites are distributed throughout the genome, so we will only be able to view a few features at a time. In this case, we will view only the first feature in the track. A convenient way to focus a view on a particular set of features is to subset the track and pass the range of the subtrack to the constructor of the view. Below we take a track subset that contains only the first feature.

```
> subTargetTrack <- targetTrack[1, ]
```

Now we call the `browserView` function to construct the view and pass the *RangesList* of the subtrack, zoomed out by a factor of 10, as the segment to view. By passing the name of the targets track in the *pack* parameter, we instruct the browser to use the "pack" mode for viewing the track. This results in the name of the microRNA appearing next to the target site glyph.

```
> view <- browserView(session, range(subTargetTrack) *
+     -10, pack = "targets")
```

Exercises

1. Create a new view with the same region as `view`, except zoomed out 2X.
2. Create a view with the "targets" track displayed in "full" mode, instead of "packed".

2.2.4 A shortcut

There is also a shortcut to the above steps. The `browseGenome` function creates a session for a specified browser, loads one or more tracks into the session and creates a view of a given genome segment. In the following code, we create a new **UCSC** session, load the track and view the first two features, all in one call:

```
> browseGenome(targetTrack, range = range(subTargetTrack) *
+     -10)
```

It is even simpler to view the subtrack in **UCSC** by relying on parameter defaults:

```
> browseGenome(subTargetTrack)
```

2.2.5 Downloading tracks

It is possible to query the browser to obtain the names of the loaded tracks and to download the tracks into R. To list the tracks loaded in the browser, enter the following:

```
> loaded_tracks <- trackNames(session)
```

One may download any of the tracks, such as the "targets" track that was loaded previously in this example.

```
> subTargetTrack <- track(session, "targets")
```

By default, the segment of the track downloaded is the current default genome segment associated with the session. One may download track data for any genome segment, such as those on a particular chromosome.

```
> posTargets <- track(session, "targets", range(chr1Targets))
```

Exercises

1. Get the SNP under the first target, displayed in `view`.
2. Get the UCSC gene for the same target.

2.2.6 Accessing view state

The `view` variable is an instance of *BrowserView*, which provides an interface for getting and setting view attributes. Note that for the UCSC browser, changing the view state opens a new view, as a new page must be opened in the web browser.

To programmatically query the segment displayed by a view, use the `range` method for a *BrowserView*.

```
> segment <- range(view)
```

Similarly, one may get and set the names of the visible tracks in the view.

```
> visible_tracks <- trackNames(view)
> trackNames(view) <- visible_tracks
```

The visibility mode (hide, dense, pack, squish, full) of the tracks may be retrieved with the `ucscTrackModes` method.

```
> modes <- ucscTrackModes(view)
```

The returned value, `modes`, is of class *UCSCTrackModes*. The modes may be accessed using the `[` function. Here, we set the mode of our "targets" track to "full" visibility.

```
> modes["targets"]
> modes["targets"] <- "full"
> ucscTrackModes(view) <- modes
```

Existing browser views for a session may be retrieved by calling the `browserViews` method on the *browserSession* instance.

```
> views <- browserViews(session)
> length(views)
```

Exercises

1. Retrieve target currently visible in the view.
2. Limit the view to display only the SNP, UCSC gene and target track.
3. Hide the UCSC gene track.

3 CPNE1 expression and HapMap SNPs

Included with the **rtracklayer** package is a track object (created by the **GGtools** package) with features from a subset of the SNPs on chromosome 20 from 60 HapMap founders in the CEU cohort. Each SNP has an associated data value indicating its association with the expression of the CPNE1 gene according to a Cochran-Armitage 1df test. The top 5000 scoring SNPs were selected for the track.

We load the track presently.

```
> library(rtracklayer)
> data(cpneTrack)
```

3.1 Loading and manipulating the track

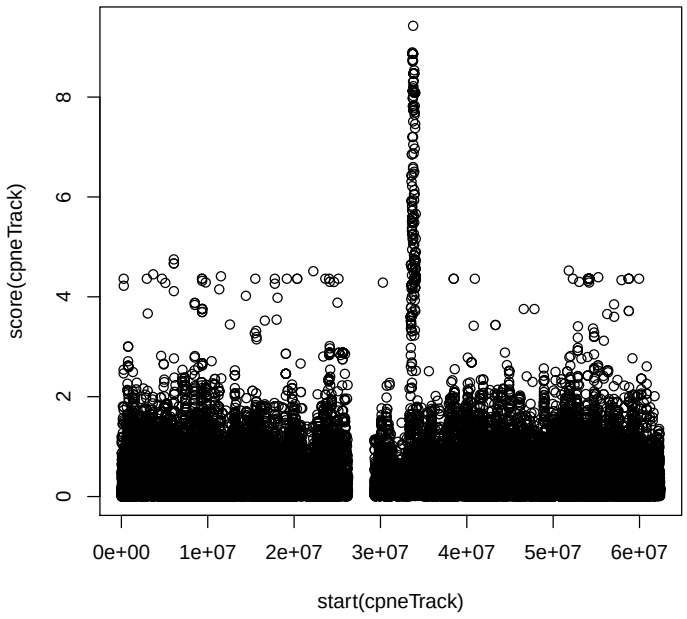
The data values for a track are stored in the columns on the *RangedData* instance. Often, a track contains a single column of numeric values, conventionally known as the *score*. The `score` function retrieves the column named or *score* or, if one does not exist, the first column in the *RangedData*, as long as it is numeric. Otherwise, `NULL` is returned.

```
> head(score(cpneTrack))
```

```
 rs4814683  rs6076506  rs6139074  rs1418258  rs7274499  rs6116610
0.16261691 0.02170423 0.47098379 0.16261691 0.05944578 0.18101862
```

One use of extracting the data values is to plot the data.

```
> plot(start(cpneTrack), score(cpneTrack))
```



3.2 Browsing the SNPs

We now aim to view some of the SNPs in the UCSC browser. Unlike the microRNA target site example above, this track has quantitative information, which requires special consideration for visualization.

3.2.1 Laying a WIG track

To view the SNP locations as a track in a genome browser, we first need to upload the track to a fresh session. In the code below, we use the `[[<-` alias of `track<-`.

```
> session <- browserSession()
> session[["cpne"]] <- cpneTrack
```

Note that because `cpneTrack` contains data values and its features do not overlap, it is uploaded to the browser in the WIG format. One limitation of the WIG format is that it is not possible to encode strand information. Thus, each strand needs to have its own track, and **rtracklayer** does this automatically, unless only one strand is represented in the track (as in this case). One could pass "bed" to the *format* parameter of `track<-` to prevent the split, but tracks uploaded as BED are much more limited compared to WIG tracks in terms of visualization options.

To form the labels for the WIG subtracks, " p" is concatenated onto the plus track and " m" onto the minus track. Features with missing track information are placed in a track named with the " na" postfix. It is important to note that the subtracks must be identified individually when, for example, downloading the track or changing track visibility.

3.2.2 Plotting the SNP track

To plot the data values for the SNP's in a track, we need to create a *browserView*. We will view the first 5 SNPs in the track, which will be displayed in the "full" mode.

```
> view <- browserView(session, range(cpneTrack[1:5, ]),
+     full = "cpne")
```

The UCSC browser will plot the data values as bars. There are several options available for tweaking the plot, as described in the help for the *wigTrackLine* class. These need to be specified laying the track, so we will lay a new track named "cpne2". First, we will turn the *autoScale* option off, so that the bars will be scaled globally, rather than locally to the current view. Then we could turn on the *yLineOnOff* option to add horizontal line that could represent some sort of cut-off. The position of the line is specified by *yLineMark*. We set it arbitrarily to the 25% quantile.

```
> track(session, "cpne2", autoScale = FALSE, yLineOnOff = TRUE,
+     yLineMark = quantile(score(cpneTrack), 0.25)) <- cpneTrack
> view <- browserView(session, range(cpneTrack[1:5, ]),
+     full = "cpne2")
```

4 Binding sites for NRSF

Another common type of genomic feature is transcription factor binding sites. Here we will use the **Biostrings** package to search for matches to the binding motif for NRSF, convert the result to a track, and display a portion of it in the **UCSC** browser.

4.1 Creating the binding site track

We will use the **Biostrings** package to search human chromosome 1 for NRSF binding sites. The binding sequence motif is assumed to be *TCAGCACCATGGACAG*, though in reality it is more variable. To perform the search, we run *matchPattern* on the positive strand of chromosome 1.

```
> library(BSgenome.Hsapiens.UCSC.hg18)
> nrsfHits <- matchPattern("TCAGCACCATGGACAG", Hsapiens[["chr1"]])
> length(nrsfHits)
```

```
[1] 2
```

We then convert the hits, stored as a *Views* object (a particular type of *IRanges* object), to a *RangedData* instance.

```
> nrsfTrack <- GenomicData(nrsfHits, strand = "+", chrom = "chr1",
+     genome = "hg18")
```

4.2 Browsing the binding sites

Now that the NRSF binding sites are stored as a track, we can upload them to the UCSC browser and view them. Below, load the track and we view the region around the first hit in a single call to `browseGenome`.

```
> session <- browseGenome(nrsfTrack, range = range(nrsfTrack[1,
+     ]) * -10)
```

We observe significant conservation across mammal species in the region of the motif.

5 Conclusion

These case studies have demonstrated a few of the most important features of **rtracklayer**. Please see the package documentation for more details.

The following is the session info that generated this vignette:

```
> sessionInfo()
```

```
R version 2.10.0 Patched (2009-10-27 r50222)
i386-apple-darwin9.8.0

locale:
[1] C/en_US.UTF-8/C/C/C/C

attached base packages:
[1] tools     stats     graphics  grDevices utils     datasets
[7] methods   base

other attached packages:
 [1] BSgenome.Hsapiens.UCSC.hg18_1.3.15
 [2] BSgenome_1.14.0
 [3] rtracklayer_1.6.0
 [4] RCurl_1.2-1
 [5] bitops_1.0-4.1
 [6] microRNA_1.4.0
 [7] Rlibstree_0.3-2
 [8] Biostrings_2.14.0
 [9] IRanges_1.4.0
[10] limma_3.2.1
[11] genefilter_1.28.0
[12] humanStemCell_0.2.3
[13] hgu133plus2.db_2.3.5
[14] org.Hs.eg.db_2.3.6
[15] RSQLite_0.7-3
[16] DBI_0.2-4
[17] AnnotationDbi_1.8.0
[18] Biobase_2.6.0

loaded via a namespace (and not attached):
[1] XML_2.6-0       annotate_1.24.0 splines_2.10.0  survival_2.35-7
[5] xtable_1.5-5
```