The rtracklayer package

Manipulating and visualizing genomic annotations

Michael Lawrence

January 20, 2009

Outline

Tracks and experimental data analysis

- Many data types have natural mapping to genome:
  - SNPs
  - Chip-seq peaks
  - Methylation
- Annotation databases contain wealth of knowledge:
  - Genes and exons (biomaRt)
  - Conservation scores
  - Transcription factor binding sites, TransFac

Tracks and experimental data analysis

- Many data types have natural mapping to genome:
  - SNPs
  - Chip-seq peaks
  - Methylation
- Annotation databases contain wealth of knowledge:
  - Genes and exons (biomaRt)
  - Conservation scores
  - Transcription factor binding sites, TransFac

Goal

Integrate the analysis of experimental data with existing annotations.

The rtracklayer package

The *rtracklayer* package is an interface (or *layer*) between **R**, genome browsers and genomic annotations.

Feature overview

- Annotation track representation and import/export (files and online databases)
- The control and querying of external genome browser sessions and views.
- Currently supports UCSC browser and database.

Case Study: Gene expression and microRNAs

Data	Microarray time course of human stem cell differentiation
Source	Tewari lab at the FHCRC
Question	Are microRNAs regulating gene expression during differentiation?
Analysis	

1. Find the differentially expressed genes
2. Create a track with microRNA target sites on DE genes
3. Upload track to genome browser to view in genomic context

Outline

Storing data on intervals
The RangedData object

- *RangedData* objects, defined by the *IRanges* package, hold data on (genomic) intervals.
- Two components
  1. The interval starts and widths, segregated by chromosome
  2. The variables describing the intervals

Preparing the data

- Used limma to find genes with changed expression after differentiation
- Obtained microRNA target sites from MiRBase, available from *microRNA* package
- Filtered the target sites for those near DE genes
- Available as dataset in *rtracklayer* package

Preparing the data

- Used limma to find genes with changed expression after differentiation
- Obtained microRNA target sites from MiRBase, available from *microRNA* package
- Filtered the target sites for those near DE genes
- Available as dataset in *rtracklayer* package

Code

```
> library(rtracklayer)
> data(targets)
```

Constructing the *RangedData* instance

1. Construct *IRanges* instance holding the endpoints of each target site
2. Construct *RangedData* with ranges, strand, chromosome and Ensembl transcript IDs

Constructing the *RangedData* instance

1. Construct *IRanges* instance holding the endpoints of each target site

Code

```
> targetRanges <- IRanges(targets$start, targets$end)
```

2. Construct *RangedData* with ranges, strand, chromosome and Ensembl transcript IDs

Constructing the *RangedData* instance

1. Construct *IRanges* instance holding the endpoints of each target site
2. Construct *RangedData* with ranges, strand, chromosome and Ensembl transcript IDs

Constructing the *RangedData* instance

1. Construct *IRanges* instance holding the endpoints of each target site
2. Construct *RangedData* with ranges, strand, chromosome and Ensembl transcript IDs

Code

```
> targetTrack <- with(targets,
+   GenomicData(targetRanges, target,
+                strand = strand,
+                chrom = chrom, genome = "hg18"))
```

Accessing built-in attributes

Each built-in feature attribute has a corresponding accessor method: `start`, `end`, `chrom`, `strand`, `genome`

Example

```
> head(start(targetTrack))

[1]   7762840  11957570  91921292
[4]  86981576  54270236 195970022
```

Exercises

1. Get the strand of each feature in the track
2. Get the genome for the track

Accessing built-in attributes

Each built-in feature attribute has a corresponding accessor method: `start`, `end`, `chrom`, `strand`, `genome`

Exercises

1. Get the strand of each feature in the track

   ```
   > head(strand(targetTrack))
   ```

   ```
   [1] + + - + - -
   Levels: - + *
   ```

2. Get the genome for the track

Accessing built-in attributes

Each built-in feature attribute has a corresponding accessor method: `start`, `end`, `chrom`, `strand`, `genome`

Exercises

1. Get the strand of each feature in the track

```
> head(strand(targetTrack))
[1] + + - + - -
Levels: - + *
```

2. Get the genome for the track

```
> genome(targetTrack)
[1] "hg18"
```

Accessing data columns

Any data column (including strand) is accessible via `$` and `[[`.

Example

```
> head(targetTrack$target)
[1] ENST00000054666 ENST00000196061
[3] ENST00000212355 ENST00000212369
[5] ENST00000234831 ENST00000235453
34507 Levels: ENST00000000233 ...
```

Exercise

Reconstruct (partially) the `targets` *data.frame*

Accessing data columns

Any data column (including strand) is accessible via $ and [[.

Example

```
> head(targetTrack$target)

[1] ENST00000054666 ENST00000196061
[3] ENST00000212355 ENST00000212369
[5] ENST00000234831 ENST00000235453
34507 Levels: ENST00000000233 ...
```

Exercise

Reconstruct (partially) the targets *data.frame*

```
> data.frame(chrom = chrom(targetTrack),
+            start = start(targetTrack),
+            end = end(targetTrack),
+            strand = strand(targetTrack))
```

Overview of *RangedData* subsetting

- Often need to subset track features and data columns
- Example: limit the amount transferred to a genome browser
- Matrix style: `track[i, j]`, where `i` is feature index and `j` is column index
- By chromosome: `track[i]`, where `i` indexes the chromosome

Subsetting examples and exercises

Examples

```
> ## get the first 10 targets
> first10 <- targetTrack[1:10,]
> ## get pos strand targets
> posTargets <- targetTrack[strand(targetTrack)=="+",]
> ## get chromosome 1 features
> chr1Targets <- targetTrack[1]
```

Exercise

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

Subsetting examples and exercises

Examples

```
> ## get the first 10 targets
> first10 <- targetTrack[1:10,]
> ## get pos strand targets
> posTargets <- targetTrack[strand(targetTrack)=="+",]
> ## get chromosome 1 features
> chr1Targets <- targetTrack[1]
```

Exercise

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

```
> chr2 <- targetTrack["2"]
> negChr2 <- chr2[strand(chr2) == "-",]
```

Overview of import/export

- Supported formats

BED	Browser Extended Display, display-oriented, native format of UCSC
WIG	Wiggle, sparse format for quantitative data
GFF	General Feature Format (versions 1, 2, and 3), general storage, popular at EBI

- Functions: `import` and `export`
- Extensible via plugin system

Import/export examples and exercises

Examples

```
> export(targetTrack, "targets.bed")
> restoredTrack <- import("targets.bed")
> ## as character vector
> targetChar <- export(targetTrack, format = "gff1")
```

Exercises

1. Output the track to a file in the “gff” format.
2. Read the track back into R.

Import/export examples and exercises

Examples

```
> export(targetTrack, "targets.bed")
> restoredTrack <- import("targets.bed")
> ## as character vector
> targetChar <- export(targetTrack, format = "gff1")
```

Exercises

1. Output the track to a file in the "gff" format.

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

2. Read the track back into R.

Import/export examples and exercises

Examples

```
> export(targetTrack, "targets.bed")
> restoredTrack <- import("targets.bed")
> ## as character vector
> targetChar <- export(targetTrack, format = "gff1")
```

Exercises

1. Output the track to a file in the “gff” format.

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

2. Read the track back into R.

   ```
   > targetGff <- import("targets.gff",
   +                          genome = "hg18")
   ```

Outline

The genome browser interface

- *rtracklayer* interfaces with the UCSC genome browser
- Easily extended to support other browsers
- Workflow
    1. Start a browser session
    2. Load one or more tracks
    3. Open one or more browser views of specific regions
    4. Possibly download interesting annotations into R

Starting a browser session

Code

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

The `session` object is a *BrowserSession* instance. With a session object, one may:

- Upload and download tracks to/from the genome browser
- Create browser views

The argument `"UCSC"` creates a session for the UCSC browser. To list all supported browsers:

Code

```
> genomeBrowsers()

[1] "UCSC"
```

Laying the target site track

Tracks may be loaded into a session with the `track<-`, `[[<-` and `$<-` functions.

Example

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

Exercise

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

Laying the target site track

Tracks may be loaded into a session with the `track<-`, `[[<-` and `$<-` functions.

Example

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

Exercise

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

```
> session$target100 <- targetTrack[1:100,]
```

Choosing a region to view

- The `range` function returns an object representing the genomic range of a track
- Assume we want to view a region around the first target site
  1. Get the range of the first feature
  2. Zoom out by a factor of 10

Choosing a region to view

- The `range` function returns an object representing the genomic range of a track
- Assume we want to view a region around the first target site
  1. Get the range of the first feature

Code
```
> region <- range(targetTrack[1,])
```

  2. Zoom out by a factor of 10

Choosing a region to view

- The `range` function returns an object representing the genomic range of a track
- Assume we want to view a region around the first target site
  1. Get the range of the first feature
  2. Zoom out by a factor of 10

Code

```
> region <- region * -10
```

Creating a view

Code

```
> view <- browserView(session, region)
```

The `view` object is a *BrowserView* instance. With a view object, one may:

- Change the currently visible region (pan/zoom)
- Change the visibility of tracks (show/hide)

Exercise

Create a new view with the same region as `view`, except zoomed out 2X.

Creating a view

Code

```
> view <- browserView(session, region)
```

The `view` object is a *BrowserView* instance. With a view object, one may:

- Change the currently visible region (pan/zoom)
- Change the visibility of tracks (show/hide)

Exercise

Create a new view with the same region as `view`, except zoomed out 2X.

```
> viewOut <- browserView(session, range(view) * -2)
```

A shortcut

All of the above in a single step:

```
> browseGenome(targetTrack,
+              range = range(targetTrack[1,]) * -10)
```

A session is started, the track is loaded and a view is created around the first target site.

Changing view range

The `range<-` function sets a new visible range on a view.

Example

```
> ## zoom in 2X
> range(view) <- range(view) * 2
```

Exercise

Shift the view to the second target site

Changing view range

The `range<-` function sets a new visible range on a view.

Example

```
> ## zoom in 2X
> range(view) <- range(view) * 2
```

Exercise

Shift the view to the second target site

```
> range(view) <- range(targetTrack[2,]) * -5
```

Changing track visibility

Tracks may be shown or hidden with the `visible<-` function.

Example

```
> ## hide the Conservation track
> visible(view)["Conservation"] <- FALSE
```

Exercise

Make the “Ensembl Genes” track visible

Changing track visibility

Tracks may be shown or hidden with the `visible<-` function.

Example

```
> ## hide the Conservation track
> visible(view)["Conservation"] <- FALSE
```

Exercise

Make the “Ensembl Genes” track visible

```
> visible(view)["Ensembl Genes"] <- TRUE
```

Overview

- Many browsers are built upon large databases
- Often want to incorporate the data into an R analysis
- For UCSC, this interacts with the table browser

Retrieving browser tracks

1. List available tracks
2. Download named track (e.g. “Conservation”) in currently viewed region

Retrieving browser tracks

1. List available tracks

Code

```
> head(trackNames(session))

         targets   Base Position
   "ct_targets"          "ruler"
Chromosome Band      STS Markers
     "cytoBand"         "stsMap"
    FISH Clones      Recomb Rate
   "fishClones"     "recombRate"
```

2. Download named track (e.g. “Conservation”) in currently viewed region

Retrieving browser tracks

1. List available tracks
2. Download named track (e.g. "Conservation") in currently viewed region

Code

```
> cons <- track(session, "Conservation")
> ## or specific region
> cons <- track(session, "Conservation",
+               range(view) * 2)
> ## shortcut
> cons <- session$Conservation
```

Outline

Beyond rtracklayer

- *rtracklayer* operates in the context of genome browsers
- Bioconductor has other sources of annotations:
  - The annotation packages
  - biomaRt

Session info

```
> sessionInfo()

R version 2.9.0 Under development (unstable) (--)
i686-pc-linux-gnu

locale:
C

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

other attached packages:
[1] rtracklayer_1.3.7 RCurl_0.91-0

loaded via a namespace (and not attached):
[1] BSgenome_1.11.9
[2] Biostrings_2.11.18
[3] IRanges_1.1.33
[4] Matrix_0.999375-17
[5] XML_1.98-1
[6] grid_2.9.0
[7] lattice_0.17-20
[8] tools_2.9.0
```