genomation - a toolkit for annotation and visualization of genomic data

Altuna Akalin
altuna.akalin@fmi.ch

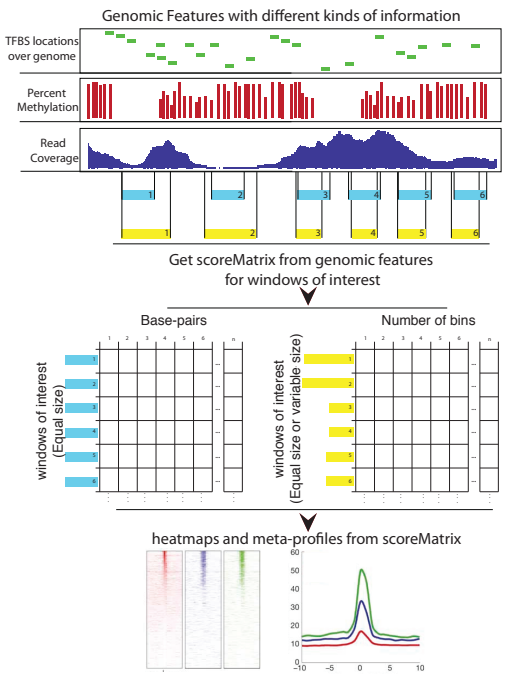
Vedran Franke
vedran.franke@gmail.com

November 11, 2013

Contents

1 Introduction

Recent advances in sequencing technologies have enabled a downpoor of biological data. The sheer amount of data has impeeded the extraction of useful knowledge and novel hypothesis generation. *genomation* is a toolkit for annotation and in bulk visualization of genomic features (scored or unscored) over predefined regions. The **genomic features** the package can handle can be anything with a minimal information of chromosome,start and end. The features could have any length and most of the time they are associated with a score. Typical examples of such data sets include aligned reads from high-throughput sequencing (HTS) experiments, percent methylation values for CpGs (or other cytosines), locations of transcription factor binding site motifs, and so on. On the other hand, throughout the vignette we use the phrase "genomic annotation" to refer to the regions of the genome associated with a potential function and do not necessarily have a score (examples: CpG islands, genes, enhancers, promoter, exons, etc.). These genomic annotations are usually the regions of interest, and distribution of genomic



features over/around the annotations are can make the way for biological interpretation of the data.

The pipeline for computational knowledge extraction consists of three steps: data filtering, integration of data from multiple sources or generation of predictive models and biological interpretation of produced models, which leads to novel hypotheses that can be tested in the wetlab. *genomation* aims to facilitate the integration of multiple sources of genomic features with genomic annotation or already published experimental results.

2 Access the data

High-throughput data which will be used to show the functonality of the *genomation* is located in two places. The annotation and cap analysis of gene expression (CAGE) data comes prepared with the genomation package, while the raw HTS data can be found in the sister package *genomationData*.
To install the *genomation* and the complementary data package the from the github repository c/p the following lines into your R interpreter:

```
library(devtools)
install_github("genomationData", username = "frenkiboy")
install_github("genomation", username = "frenkiboy")
```

The *genomationData* vignette contains a verbose description of contained files.
To list the available data:

```
list.files(system.file("extdata", package = "genomationData"))
```

To see the descriptions of the files:

```
sampleInfo <- read.table(system.file("extdata/SamplesInfo.txt",
    package = "genomationData"), header = T, sep = "\t")
head(sampleInfo)
```

Basic annotation data and processed experimental data can be found within the *genomation* package. The data can be accesed throught the data command or located in the extdata folder.

```
library(genomation)
data(cage)
data(cpgi)

list.files(system.file("extdata", package = "genomation"))
```

3 Data input

One of larger hindrances in computational genomics stems from the myriad of formats that are used to store the data. Although some formats have been selected as de facto standards for specific kind of biological data (e.g. BAM, VCF), almost all publications come with supplementary tables that do not have the same structure, but hold similar information. The tables usually have a tabular format, contain the location of elements in genomic coordinates and various metadata colums. (genomation) contais functions to read genomic features and genomic annotation provided they are in a tabular format. These functions will read the data from flat files into GRanges or GRangesList objects.

readGeneric is the workhorse of the genomation package. It is a function developed specifically for input of genomic data in tabular formats, and their conversion to a GRanges object. By default, the function persumes that the file is a standard .bed file containing columns chr, start, end.

```
library(genomation)
tab.file1 <- system.file("extdata/tab1.bed", package = "genomation")
readGeneric(tab.file1)
```

If the file contains meta data columns (as in extended bed format), it is possible to read all or some of the additional columns. To select columns which you want to read in, use the meta.col argument

```
readGeneric(tab.file1, keep.all.metadata = TRUE)

readGeneric(tab.file1, meta.col = list(CpGnum = 4))
```

If the file contains header, the function will automatically recognize the columns using the header names.

```
readGeneric(tab.file1, header = TRUE, keep.all.metadata = TRUE)
```

If the files have permutted columns, such that the first three do not represent chromosome, start and end, you can select an arbitrary set of columns using the chr, start and end arguments.

```
tab.file2 <- system.file("extdata/tab2.bed", package = "genomation")
readGeneric(tab.file2, chr = 3, start = 2, end = 1)
```

readGeneric] function can easily be extended to read almost any kind of biological data. As an example we have provided convenience functions to read the Encode narrowPeak and broadPeak formats, and gtf formatted files.

```
gff.file <- system.file("extdata/chr21.refseq.hg19.gtf",
    package = "genomation")
gff <- gffToGRanges(gff.file)
head(gff)
```

In order to split the last column of the gff file, use the split.group=TRUE argument.

```
gff <- gffToGRanges(gff.file, split.group = TRUE)
head(gff)
```

There are specific functions to read genomic annotation from flat bed files. readFeatureFlank is a convenience function used to get the ranges flanking the set of interest. As an example, it could be used to get the CpG island shores, which have been shown to harbour contidion specific differential methylation

```
# reading genes stored as a BED files
cpgi.file <- system.file("extdata/chr21.CpGi.hg19.bed",
    package = "genomation")
cpgi.flanks <- readFeatureFlank(cpgi.file)
head(cpgi.flanks$flanks)
```

```
readGeneFeatures()
```

4 Extraction and visualization of genomic data

A standard step in a computational genomics experiment is the visualization of average enrichment over a certain predefined set of ranges, such as a visualization of mean coverage of a certain histone modification around a transcription factor binding site, or visualization of histone positions around transcription start sites. *genomation* provides convenience functions for extraction and visualization of data over predefined windows.

4.1 Extration of data over predefined winows

ScoreMatrix and ScoreMatrixBin are functions used to extract data over predefined windows. ScoreMatrix is used when all of the windows have the same width, such as an area around the transcription start site, while the ScoreMatrixBin is designed for use with windows of unequal width (e.g. enrichment of methylation over exons). Both functions have 2 main arguments: target and windows. target is the data that we want to extract, while the windows represents the regions over which we want to see the enrichment. The target data can be in 3 forms: a GRanges, a RLeList or a path to an indexed .bam file. The windows must be GRanges object.

As an example we will extract the density of cage tags around the promoters on the human chromosome 21.

```
data(cage)
data(promoters)
sm <- ScoreMatrix(target = cage, windows = promoters)
sm
```

ScoreMatrixBin function has an additional bin.num argument which specifies by how many bins will each window be represented (ie. it converts windows of unequal width into ones of equal width.). By default, the binning function is set to mean.

```
data(cage)
gff.file <- system.file("extdata/chr21.refseq.hg19.gtf",
    package = "genomation")
exons <- gffToGRanges(gff.file, filter = "exon")
sm <- ScoreMatrixBin(target = cage, windows = exons,
    bin.num = 50)
```

As you can see, running ScoreMatrixBin with bin.num=50 on a set of exons warned us that some of the ranges had width less than 50 bp and were removed from the set.

To simultaneously work on multiple files you can use the ScoreMatrixList function. The function likewise has 2 obligatory arguments targets and windows. While the windows is the same as in ScoreMatrix, the targets argument is contains results from multiple experiments. It can be in one of the three formats: a list of RleLists, a list of GRanges (or a GRangesList object), or a character vector designating a set of .bam files.

```
data(promoters)
data(cpgi)
data(cage)

cage$tpm <- NULL
targets <- list(cage = cage, cpgi = cpgi)
sm <- ScoreMatrixList(targets = targets, windows = promoters,
    bin.num = 50)
```

If all of the windows have the same width ScoreMatrixList will use the ScoreMatrix function. That can be overridden by explicity specifying the bin.num argument, as we did in the example.

4.2 Visualization of multiple genomic experiments

There are 2 basic modes of visualization of enrichment over windows: either as a heatmap, or as a line grap. heatMatrix], plotMeta and multiHeatMatrix are functions for visualization of ScoreMatrix and ScoreMatrixList objects.

We will plot the distribution of CAGE tags around promoters on human chr21.

```
## Warning:  data set 'cage' not found
## Warning:  data set 'promoters' not found
## Error:  could not find function "ScoreMatrix"
## Error:  could not find function "heatMatrix"
## Error:  could not find function "plotMeta"
```

The heatMatrix function can also take a list of numeric vectors designating row names, or a factor variable that represent our annotation over the windows.

```
## Warning:  data set 'cage' not found
## Warning:  data set 'promoters' not found
## Warning:  data set 'cpgi' not found
## Error:  could not find function "ScoreMatrix"
## Error:  could not find function "countOverlaps"
## Error:  could not find function "countOverlaps"
## Error:  could not find function "heatMatrix"
```

5 Annotation of genomic features

6 Use cases for genomation package

The genomation package provides generalizable functions for genomic data analysis and visualization. Below we will demonstrate the functionality on specific use cases

6.1 Annotation of HTS data by functional regions

6.2 Visualization of ChiP sequencing data

We will visualize the binding profiles of 6 transcription factors around the Ctcf binding sites.

In the fist step we will select the *.bam files containing mapped reads.

```
## Loading required package:  BiocGenerics
## Loading required package:  parallel
##
## Attaching package:  'BiocGenerics'
##
## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB,
##    clusterCall, clusterEvalQ,
##    clusterExport, clusterMap, parApply,
##    parCapply, parLapply, parLapplyLB,
##    parRapply, parSapply, parSapplyLB
##
## The following object is masked from 'package:stats':
##
```

```
## Error:  could not find function "ScoreMatrixList"
## Error:  error in evaluating the argument 'x' in selecting a method for function
'match':  Error:  object 'sml' not found
## Error:  could not find function "multiHeatMatrix"
```

Figure 2: Heatmap profile of unscaled coverage shows a slight colocalization of Ctcf, Rad21 and Znf143; option `fig.keep='last'` here.

```
## Error:  could not find function "scaleScoreMatrixList"
## Error:  could not find function "multiHeatMatrix"
```

Figure 3: Heatmap profile of scaled coverage shows much stronger colocalization of the transcription factors; nevertheless, it is evident that some of the CTCF peaks have a very weak enrichment. option `fig.keep='last'` here.

```
##    xtabs
##
## The following objects are masked from 'package:base':
##
##    anyDuplicated, append, as.data.frame,
##    as.vector, cbind, colnames, duplicated,
##    eval, evalq, Filter, Find, get,
##    intersect, is.unsorted, lapply, Map,
##    mapply, match, mget, order, paste, pmax,
##    pmax.int, pmin, pmin.int, Position,
##    rank, rbind, Reduce, rep.int, rownames,
##    sapply, setdiff, sort, table, tapply,
##    union, unique, unlist
##
## Loading required package:  IRanges
## Loading required package:  XVector
```

Firstly, we will read in the Ctcf peaks, filter regions from human chromosome 21, and order them by their signal values. In the end we will resize all ranges to have a uniform width of 500 bases, fixed on the center of the peak.

```
## Error:  could not find function "readBroadPeak"
## Error:  object 'ctcf.peaks' not found
## Error:  object 'ctcf.peaks' not found
## Error:  error in evaluating the argument 'x' in selecting a method for function 'resize':
Error:  object 'ctcf.peaks' not found
```

In order to extract the coverage values of all transcription factors around chipseq peaks, we will use the ScoreMatrixList function. ScoreMatrixList assign names to each element of the list based on the names of the bam files. We will use the names of the files to find the corresponding names of each sample in the SamplesInfo.txt Using the heatmapProfile on our ScoreMatrixList, we can plot the underlying signal side by side.

Because of the large range of signal values in chipseq peaks, the heatmapProfile will not show the true extent of colocalization. To get around this, it is advisable to independently scale the rows of each element in the ScoreMatrixList.

```
## Error:  object 'tf.comb' not found
## Error:  could not find function "ScoreMatrixList"
## Error:  error in evaluating the argument 'x' in selecting a method for function
'match':  Error:  object 'sml' not found
## Error:  could not find function "scaleScoreMatrixList"
## Error:  could not find function "multiHeatMatrix"
```

6.3 Combinatorial binding of transcription factors

In the first step we will read all peak files into a GRanges list. We will use the SamplesInfo file from the *genomationData* to get he names of the samples. Four of the peak files are in the Encode broadPeak format, while one is in the narrowPeak. To read the files, we will use the readGeneric function. It enables us to select from the files only columns of interest. As the last step, we will restrict ourselves to peaks that are located on chromosome 21 and have width 100 and 1000 bp

```
## Ctcf
## Error:  could not find function "readGeneric"
## Error:  Could not find a 'CompressedList' subclass for values of type 'NULLOptionalFunctionOptionalMe
## Error:  unable to find an inherited method for function 'seqnames' for signature '"list"'
```

To find the combination of binding sites we will use the findFeatureComb function. It takes a granges list object, finds the union of the ranges and designates each range by the combination of overlaps from the original set. By default, the returned ranges will have a numeric class meta data column, which designates the correponding combination. If you are interested in the names of the TF which make the combinations, put the use.names=TRUE.

```
## Error:  could not find function "findFeatureComb"
```

To visualize the results, we will plot the enricment of resulting regions. Before doing that we will order the regions by their class argument. The plot shows perfectly how misleading the peak calling process can be. Although the plots show that CTFC, Rad21 and Znf143 have almost perfect colozalization, Peak callers have trouble identifying peaks in regions with lower enrichments and as a result, we get different statistics when using overlaps.

6.4 Annotation of bam files