Goldmine: Integrating information to place epigenome-wide results into biological contexts

Jeff Bhasin

2015-05-08

Contents

Purpose

This how-to will demonstrate the use of Goldmine to analyze a set of example genomic ranges in order to introduce you to the main functions of the package. These ranges can be substituted for any ranges of interest. Please refer to the function documentation for advanced options.

Prerequisites

- R Installed (download from http://cran.r-project.org/)
- For large sets of ranges and for optimal performance, we recommend using Goldmine on a linux server with at least 8GB of RAM. However, it will function on desktop computers for smaller sets of ranges.

Installation

First, start R and install pre-requisite packages from Bioconductor:

```
source("http://bioconductor.org/biocLite.R")
biocLite(c("GenomicRanges","IRanges","devtools"))
```

Then, install Goldmine from GitHub. Be sure to accept installation of any additional pre-requisite packages from CRAN.

```
library(devtools)
install_github("jeffbhasin/goldmine")
```

For the enrichment example, the BSgenome of hg19 is also required, but this is not required for Goldmine as a whole:

```
source("http://bioconductor.org/biocLite.R")
biocLite("BSgenome.Hsapiens.UCSC.hg19")
```

Loading Genomic Ranges

After Goldmine is installed, it must be loaded before the functions will be available to a session of R:

```
library(goldmine)
```

The goldmine package contains an example set of genomic ranges. These ranges are a pre-filtered set of differentially methylated regions (DMRs) detected between CD4+ and CD8+ T-cells that were detected using Methylaction on MeDIP-seq data produced by the Epigenome Roadmap Constortium.

To load the example genomic ranges from a CSV file:

```
csvpath <- system.file("extdata", "dmrs.csv", package = "goldmine")
query <- read.csv(csvpath)
```

If you are providing your own set of ranges, be sure the data contains the columns “chr”, “start”, and “end” which represent chromosome name, start coordinate (1-based), and end coordinate, respectively.

```
head(query)
```

Annotation of Genomic Ranges

Both summary (“wide” format) and detailed (“long” format) annotations are produced by the goldmine() function. The data source for the gene and feature sets is the table archive of the UCSC Genome Browser. The first time a table is needed, Goldmine will download and cache the table. On subsequent calls to goldmine(), and other functions that access UCSC Genome Browser tables, the data will only be re-downloaded if there has been an update to the table on UCSC’s server. It is the user’s responsibility to ensure their use of this external resource meets UCSC’s Conditions of Use.

To enable UCSC table caching, please choose a cache directory. This is a folder on your computer that Goldmine will use to download and cache the reference data used for the annotations. For the purposes of this example, we will use a folder called "gbcache" in the current working directory. Please set the value of the cachedir variable to point to your cache directory of choice.

```
cachedir <- "gbcache"
```

Goldmine supports all of the assembled genomes on the UCSC Genome Browser that have either UCSC (knownGene), RefSeq (refGene), or ENSEMBL (ensGene) gene annotation tables available. In the case of the DMR data, the genomic coordinates are with respect to the hg19 build of the human genome. If using your own ranges, please set the genome variable to match the UCSC assembly name of the correct genome (e.g. "mm10", "mm9", "hg18", etc).

```
genome <- "hg19"
```

For speed of the example, you can subset to the first 100 DMRs only. Feel free to skip this if you are on a powerful computer. We recommend using the full set of DMRs if possible for the section on enrichment analysis in order to produce meaningful genome-wide results.

```
nrow(query)
query <- query[1:100,]
nrow(query)
```

With these two variables set, the goldmine() function can now be run. If this is your first run or first time using a cache directory, it may take a few minutes to download the reference genome browser tables.

```
gm <- goldmine(query=query,genome=genome,cachedir=cachedir)
```

The output object (in this case we called it "gm") is an R list with multiple elements. Let's look at each one individually.

```
summary(gm)
```

The first element is "context", which is a "wide" format annotation of the query ranges. It will have the same number of rows as the query, and reports them in the same order as the query. All columns of the query are retained, and additional columns are added to summarize the genomic context with respect to gene models.

```
nrow(gm$context)
colnames(gm$context)
```

However, gene annotations can be very complex due to overlapping/nested genes and the diversity of gene isoforms. To capture isoform-level detail, the "gene" table is generated. This is a "long" format table, which is similar to an inner join in SQL, contains a row for each pair of overlaps between a query range and an entry in the gene database. Thus, there will be a row for each individual gene isoform overlapped by each query range, and there will be columns to describe which portions of the gene model are overlapped.

```
nrow(gm$genes)
colnames(gm$genes)
```

The final element of the list is "features", which in this run is currently empty because no feature sets were specified. See the next section for how to add feature annotations.

By default, the UCSC knownGene table is used to provide the gene database. In practice, we have found this gene set to be a good compromise between the manually-curated RefSeq and the more prediction-inclusive Ensembl genes. Goldmine provides the getGenes() function to load the genes from any of these gene sets, and the "genes" option to the goldmine() function allows using any custom list of genes. This could also be used, for example, to only annotate using a subset of one of the gene databases. In this case, we will restrict to only coding genes from RefSeq.

```
genes <- getGenes("refseq",genome=genome,cachedir=cachedir)
genes <- genes[str_detect(genes$isoform.id,"NM"),]
gm <- goldmine(query=query,genes=genes,genome=genome,cachedir=cachedir)
nrow(gm$genes)
```

Annotation of Features

In addition to gene models, Goldmine can report annotation and overlap with any feature set available from UCSC. Please see the UCSC Table Browser to browse all tables by category for a given genome. The "describe table schema" button can provide useful descriptions of the tables.

For this example, we will annotate with features of common interest to many epigenome-wide experiments: ENCODE ChIP-seq peaks, ENCODE DNaseI hypersensitive sits, and CpG islands/shores/shelves. The ENCODE datasets can be obtained using the getFeatures() function and the special function getCpgFeatures() can automatically generate CpG island/shore/shelve features for any genome with a "CpgIslandsExt" table available. If you have your own feature sets, they can also be included. Make sure they include the columns "chr", "start", and "end".

```
features <- getFeatures(tables=c("wgEncodeRegDnaseClusteredV3",
                         "wgEncodeRegTfbsClusteredV3"),
                         genome=genome, cachedir=cachedir)
features <- c(features,getCpgFeatures(genome=genome,cachedir=cachedir))
```

```
summary(features)
```

When all feature sets of interest have been joined into an R list object, this object can be provided to the "features" option of goldmine() and annotation performed.

```
gm <- goldmine(query=query,features=features,genome=genome,cachedir=cachedir)
```

This will change the output in two ways. First, under the "context" table, there will be new columns, one for each feature set, representing the percent overlap of the query range with ranges from the feature set. Also, a detailed accounting of each overlap in "long" format is available in the "features" list. This sub-list contains a table for each feature set, and contains one row for each pair of query to feature overlaps. It includes all columns from the feature tables, so that more specific details about each feature (i.e. factor name, experiment IDs, peak scores, etc) can be examined.

```
colnames(gm$context)
summary(gm$features)
colnames(gm$features$wgEncodeRegTfbsClusteredV3)
```

The gmWrite() function simplifies saving all tables in an output list from goldmine() as CSV files for viewing in a spreadsheet or processing outside of R.

```
gmWrite(gm, path="gm_csv")
```

Enrichment of Features

Some sets of features, such as the ENCODE ChIP-seq supertrack, contain many different types of features. In the case of the ChIP-seq data, there are experimentally-derived binding sites for many different transcription factors represented. A logical question is, are any of these specific factors enriched in the query ranges when compared to a null expectation?

Goldmine provides a mechanism to generate appropriate null sequence sets and test for enrichment of features using a binomial test. In particular, Goldmine can match the null set for multiple sequence covariates, including length, CpG density, GC%, and repeat content.

An enrichment analysis consists of three steps:

1. Obtain a background pool of sequences from which to select the null set
2. Match for length, and optionally, other variables to select a null set from the pool
3. Count the occurrence of each feature in the query set and the null set and compute enrichment

Background Pool of Sequences

The background pool of sequences can either be selected for a particular analysis, for example, non-differential regions from a targeted assay, or can be drawn from the genome at large. Goldmine provides a function, drawGenomePool() to draw a length-matched pool of sequences from the genome. This function requires that pool sequences do not overlap with the query or each other, do not run off chromosome ends, and do not occur in assembly gap regions as defined by the "gap" track. The pool should be made a number of times larger than the query using the "n" option. In practice, selection of "n" will need to be calibrated for the query range set. If there is a very large set with a large spread of lengths, then it may not be possible to draw very large pools for certain cases. We recommend using as large of an "n" as possible, so that the matching function has the largest pool of sequences to match from. We recommending setting and saving the seed for the random number generator so that the draw can be reproduced.

```
set.seed(999)
pool.gr <- drawGenomePool(query=query,n=10,
                          chrs=paste0("chr",c(1:22,"X","Y")),
                          genome=genome,cachedir=cachedir)
length(pool.gr)
```

Note that the "chrs" option is highly recommended. If not set, the function will restrict to only the chromosomes names present in query. We recommending setting "chrs" to the names of all the canonical chromosomes (autosomes and sex chromosomes) in your genome. For the human genome, this would be chr1 through chr22, chrX, and chrY (as set in the example above).

Matched Null Set

Goldmine uses propensity score matching to extract a sub-set from the background pool that is matched for multiple sequence covariates. For this example, we will match based on length and CpG density. In order to calculate these sequence covariates, Goldmine needs access to the sequence of the genome in question. These sequences can be accessed via the BSgenome packages in Bioconductor. In this case, we will need to load the hg19 human genome. The performance of the matching is plotted in a series of histograms, QQ plots, and

variable plots which will be saved at the given output directory path. FASTA format files of both the query set and null set are also saved in this directory, and be used for motif analysis in programs that depend on appropriate null sets such as RSAT and DREME.

```
library(BSgenome.Hsapiens.UCSC.hg19)
bsg <- BSgenome.Hsapiens.UCSC.hg19
formula <- "treat~sizeLog+freqCpG"
psm <- doPropMatch(query=query, pool=pool.gr, n=1, formula=formula,
                   outdir="gm_match",
                   bsg=bsg,
                   genome=genome, cachedir=cachedir)
```

Test for Enrichment

With a null set generated, the null ranges can be used as a background set for a test of enrichment.

First, we must select a set of features to test enrichment for. In this example, we will test for the ENCODE ChIP-seq peaks and TRANSFAC predicted motifs. Feature sets must be given as a data.frame, and have a column called "name" which specifies each category of feature within the feature set. In this case, "name" will be the name of the ChIP-seq protein or the ID of the predicted motif from TRANSFAC. In the case of these two examples, this column already exists in the tables output by getFeatures().

```
features <- getFeatures("wgEncodeRegTfbsClusteredV3",genome,cachedir)[[1]]
```

```
te_enc <- testEnrichment(query=query, null=psm$ranges.null, features=features)
```

The output then contains statistics for the enrichment.

```
colnames(te_enc)
```

The test can also be performed for predicted binding sites from TRANSFAC.

```
features <- getFeatures("tfbsConsSites",genome,cachedir)[[1]]
te_trans <- testEnrichment(query=query, null=psm$ranges.null, features=features)
```

The output from a call to testEnrichment() can be saved using R's built in write.csv() function for viewing in a spreadsheet program or processing outside of R.

```
write.csv(te_enc,row.names=F,file="te_enc.csv")
write.csv(te_trans,row.names=F,file="te_trans.csv")
```

Appendix

Direct Import of UCSC Genome Browser Tables

The goldmine(), getGenes(), and getFeatures() functions all call the getUCSCTable() function. This function handles the download and caching of data from the UCSC FTP server. This function can also be used directly for custom analysis that requires easy access to these useful tables. Note that start coordinates in the raw tables are 0-based. All Goldmine output has been adjusted to be 1-based, except in the case of raw table data from getUCSCTable().

For example, we could download ENCODE CTCF ChIP-seq data for the cell line HCT116:

```
tab <- getUCSCTable(table="wgEncodeAwgTfbsUwHct116CtcfUniPk",
                        genome=genome, cachedir=cachedir)
```

Please see the UCSC Table Browser to obtain table names and schema.

Reproducible Annotation

By default, Goldmine will ensure that the latest versions of reference tables from the UCSC Genome Browser are obtained. This is accomplished by comparing the date of the version in the cache to the date of the version on UCSC's server, and if UCSC's version is newer, the new version will be downloaded and used. To ensure reproducibility, versions can be frozen by setting the option sync=FALSE in the goldmine(), getGenes(), and getFeatures() functions. We recommend setting cachedir to a project-specific location, rather than a common location where other calls to goldmine() might download new versions of the data. Then, by setting sync=FALSE, the latest version will be downloaded the first time the script is run, and new versions will not be downloaded or checked for on subsequent runs. This ensures that the reference tables are static for a given project, so the annotation can be reproduced.