

Identification of TF binding profiles from ChIP-seq and DNase-seq using SeqGL

Manu Setty¹, Christina Leslie

Computational Biology Program, MSKCC,
New York

¹manu@cbio.mskcc.org

May 29, 2014

Abstract

SeqGL is a new group lasso-based algorithm to extract multiple transcription factor (TF) binding signals from ChIP- and DNase-seq profiles. Benchmarked on over 100 ChIP-seq experiments, SeqGL outperformed traditional motif discovery tools in discriminative accuracy and cofactor detection. SeqGL successfully scales to DNase-seq data, identifying a large multiplicity of TF signals confirmed by ChIP, and can be used with multitask training to learn genomic-context and cell-type specific TF signals.

Contents

1	Installation	1
1.1	Dependencies	1
2	Inputs	2
3	SeqGL wrapper	2
4	SeqGL details	4
4.1	Getting data ready for SeqGL	4
4.2	Identification of groups	5
4.3	Group lasso	6

1 Installation

The R package can be downloaded from <https://bitbucket.org/leslielab/seqgl>.

1.1 Dependencies

- The following bioconductor packages: Biostrings, GenomicRanges, BSgenome, WGCNA, fastcluster, gtools, sfsmisc, kernlab. These packages can be installed using the command

```
source("http://bioconductor.org/biocLite.R")
biocLite(c("Biostrings", "GenomicRanges", "BSgenome", "WGCNA", "fastcluster",
           "gtools", "sfsmisc", "kernlab"))
```

- ChIPKernels package for wildcard kernel. This package can be downloaded from <https://bitbucket.org/leslielab/chipkernels>.

- spams toolbox for running group lasso. Download and install the R package from <http://spams-devel.gforge.inria.fr/downloads.html>.
- HOMER motif finding tool for associating groups with motifs. <http://biowhat.ucsd.edu/homer/ngs/index.html>.
- BSgenome package for the organisms of your choice from Bioconductor. Example if the peaks are from hg19, install BSgenome.Hsapiens.UCSC.hg19 from <http://bioconductor.org/packages/release/data/annotation/html/BSgenome.Hsapiens.UCSC.hg19.html>. The example detailed in this vignette assumes this package has been installed.

After all the dependencies are installed, SeqGL can be installed from source using R CMD INSTALL <path to package>.

2 Inputs

Chip-seq or DNase-seq peaks are the inputs to SeqGL. The peaks should be provided in bed format and should contain the following columns.

- chrom: Chromosome
- start: Genomic start
- end: Genomic end
- strand: Strand
- score: Score to rank the peaks. Can be $-\log(p - value)$.
- summit: Summit position in the peak
- name: Unique identifier for each peak

An example peaks file can be found in the package. This bed file contains the top peaks for IRF4 ChIP-seq in GM12878, a lymphoblastoid cell line.

```
peaks.file <- system.file("extdata/gm12878_top_irf4_peaks.bed", package = "SeqGL")
peaks <- read.table(peaks.file, header = TRUE)
head(peaks)
```

```
##   chrom chromStart  chromEnd strand  score  summit  name
## 1  chr1   37754504  37754786      *   2089    149 Peak1
## 2  chr1  181068642 181068903      *    916    136 Peak2
## 3  chr1   45965797  45966083      *   1574    162 Peak3
## 4  chr1  163291458 163291723      *   1876    107 Peak4
## 5  chr1  235114142 235114503      *   1754    127 Peak5
## 6  chr1   28526489  28526779      *   1525    160 Peak6
```

3 SeqGL wrapper

SeqGL has a wrapper function `run.seqGL` which takes the peaks, organism as inputs and run through the complete pipeline.

```
peaks.file <- system.file("extdata/gm12878_top_irf4_peaks.bed", package = "SeqGL")
run.seqGL(peaks.file, out.dir = "seqGL.Test/", data.type = "ChIP", org = "hg19")
```

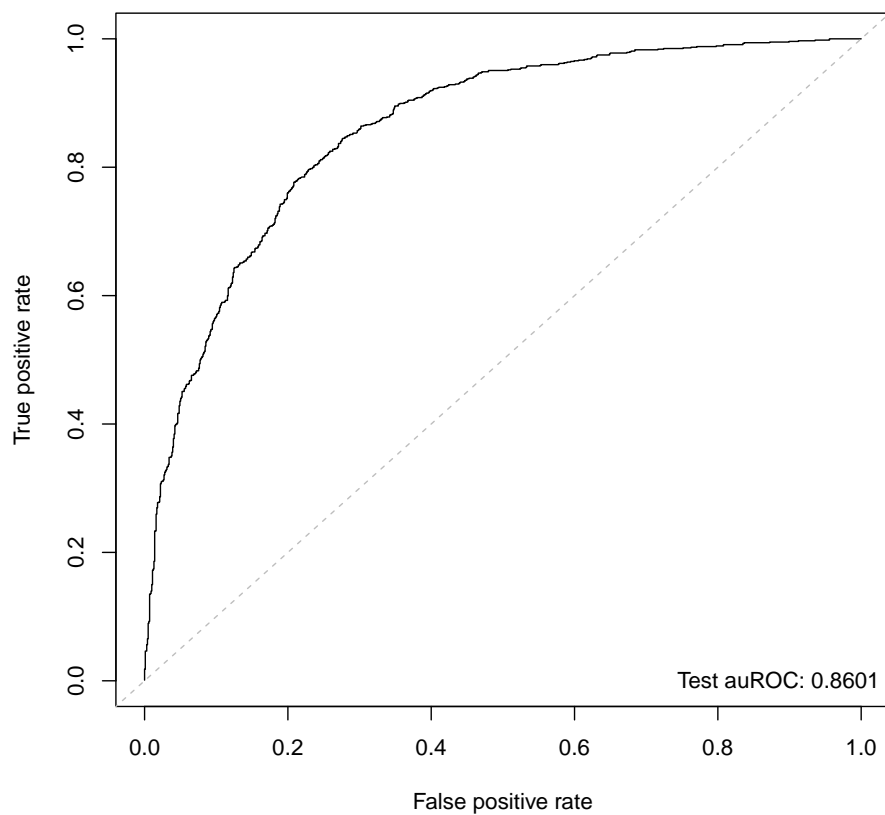


Figure 1: **Test auROC for IRF4 peaks**

The results of the will be present in the `seqGL.Test` folder and contains the following objects. The test performance is shown in the file `seqGL.Test/test_auc.pdf` which shows the ROC plot. See Figure 1 for an example.

The motifs associated with each group can be found in `seqGL.Test/group_motifs.html`. A screenshot is shown in Figure 2.





Group	Test Class	Score	Motif	Motif TF
Group2	1	199.45		HIF1b(HLH)
Group14	1	163.23		PU.1-IRF
Group20	1	95.76		PU.1(ETS)
Group5	1	38.71		RUNX2(Runt)

Figure 2: **Group scores and motifs for IRF peaks**

The contents of the results folder:

- group_motifs.html: Html file containing the group scores and motif associated with each group.
- test_auc.pdf: ROC plot showing the performance of the method. auROC varies from 0.5 – 1 with being perfect classification.
- group_members: Folder containing peaks associated with each group.
- group_motifs: Folder containing all the HOMER results.
- train_test_data.Rds R object containing the train and test data.
- clustering_results.Rds R object containing clustering results.
- group_lasso_results.Rdata R object containing all the group lasso results.

4 SeqGL details

This section describes the different steps underlying SeqGL using the IRF4 peaks as an example. The following command will load the library.

4.1 Getting data ready for SeqGL

The first step is to normalize the spans of different peaks and ensure they are all of the same width. We recommend a span of 150 bases around the peak summit for optimal performance.

```
peaks.gr <- GRanges(peaks$chrom, IRanges(peaks$chromStart, peaks$chromEnd),
  summit = peaks$summit, name = peaks$name)
span <- 150
pos.regions <- peaks.gr
start(pos.regions) <- start(pos.regions) + pos.regions$summit - 1
end(pos.regions) <- start(pos.regions)
pos.regions <- resize(pos.regions, span, fix = "center")
```

Then create positive and negative regions. Negative regions are created by shifting the positive regions upstream.

```
neg.regions <- shift(pos.regions, span * 2)
```

After creating the regions or examples, we build the feature matrices for group lasso. The `build.features.kernels` function from `ChIPKernels` package is used for constructing the feature matrices. Wildcard string kernels are used for determining feature matrices. The `build.train.test.data` function splits the examples into train and test sets, determines sequences for all the examples and then builds the feature matrices. `BSgenome` package corresponding to the organisms should be installed for determining sequences. Specifically, the peaks are for hg19 genome and `BSgenome.Hsapiens.UCSC.hg19` has to be installed in this example.

```
res.dir <- "/tmp/SeqGLTest"
dir.create(res.dir)
dictionary.file <- system.file("extdata/wildcard_dict_kmer8_mismatches2_alpha5_consecutive_mis.Rdata",
  package = "SeqGL")
train.test.data <- build.train.test.data(pos.regions, neg.regions, dictionary.file,
  org = "hg19")

## [1] "Extract sequences for all examples..."
## [1] "Time for extracting sequences: 0.93 minutes"
## [1] "Building features from dictionary..."
## [1] "Position 1 of 143"
## [1] "Position 21 of 143"
## [1] "Position 41 of 143"
## [1] "Position 61 of 143"
## [1] "Position 81 of 143"
## [1] "Position 101 of 143"
## [1] "Position 121 of 143"
## [1] "Position 141 of 143"
## [1] "Time for determining features : 1.48"
## [1] "Selecting top features..."
## [1] "Time for selecting features: 0.38 minutes"
## [1] "Package and return..."
## [1] "Total time for constructing data: 2.83 minutes"

saveRDS(train.test.data, file = sprintf("%s/train_test_data.Rds", res.dir))
```

`train.test.data` is a list containing all the training and test data

```
show(labels(train.test.data))

## [1] "train.features" "test.features" "train.inds"
## [4] "test.inds"      "feature.inds"  "train.labels"
## [7] "test.labels"    "train.regions" "test.regions"
## [10] "dictionary.file" "train.seqs"    "test.seqs"
```

4.2 Identification of groups

The groups are identified by hierarchical clustering of features. `run.clustering` function to used for hierarchical clustering.

```
clustering.results <- run.clustering(train.test.data$train.features, no.groups = 20)

## [1] "Running clustering..."
## [1] "Time for running clustering: 2.47 minutes"

saveRDS(clustering.results, file = sprintf("%s/clustering_results.Rds", res.dir))
```

4.3 Group lasso

The groups of kmers are used in a group lasso learning framework. We use the `spams` toolbox to run group lasso. We first identify the optimal regularization parameters for group lasso and learn the model using these parameters. The functions `group.lasso.eval.parameters` and `run.group.lasso` are used for parameter evaluation and running group lasso respectively.

```
lambdas = c(0.01, 0.005, 0.001, 5e-04, 1e-04)
param.eval <- group.lasso.eval.parameters(train.test.data$train.features, train.test.data$train.labels,
    train.test.data$test.features, train.test.data$test.labels, clustering.results$groups,
    lambdas = lambdas)

## [1] "Running crossvalidation ..."
## [1] "i: 1 of 5, j: 1 of 5"
## [1] "i: 1 of 5, j: 2 of 5"
## [1] "i: 1 of 5, j: 3 of 5"
## [1] "i: 1 of 5, j: 4 of 5"
## [1] "i: 1 of 5, j: 5 of 5"
## [1] "i: 2 of 5, j: 1 of 5"
## [1] "i: 2 of 5, j: 2 of 5"
## [1] "i: 2 of 5, j: 3 of 5"
## [1] "i: 2 of 5, j: 4 of 5"
## [1] "i: 2 of 5, j: 5 of 5"
## [1] "i: 3 of 5, j: 1 of 5"
## [1] "i: 3 of 5, j: 2 of 5"
## [1] "i: 3 of 5, j: 3 of 5"
## [1] "i: 3 of 5, j: 4 of 5"
## [1] "i: 3 of 5, j: 5 of 5"
## [1] "i: 4 of 5, j: 1 of 5"
## [1] "i: 4 of 5, j: 2 of 5"
## [1] "i: 4 of 5, j: 3 of 5"
## [1] "i: 4 of 5, j: 4 of 5"
## [1] "i: 4 of 5, j: 5 of 5"
## [1] "i: 5 of 5, j: 1 of 5"
## [1] "i: 5 of 5, j: 2 of 5"
## [1] "i: 5 of 5, j: 3 of 5"
## [1] "i: 5 of 5, j: 4 of 5"
## [1] "i: 5 of 5, j: 5 of 5"
## [1] "Time for running crossvalidation: 2.00 minutes"

saveRDS(param.eval, file = sprintf("%s/param_eval.Rds", res.dir))

inds <- which(param.eval$aucs.matrix == max(param.eval$aucs.matrix), arr.ind = TRUE)
group.lasso.results <- run.group.lasso(train.test.data$train.features, train.test.data$train.labels,
    train.test.data$test.features, train.test.data$test.labels, clustering.results$groups,
    param.eval$lambdas[inds[1]], param.eval$lambdas[inds[2]])

## [1] "Running group lasso..."
## [1] "Time for running group lasso: 0.08 minutes"
```

Group rankings and peaks associated with group can be determined using

```
group.error.changes <- determine.group.error.changes(train.test.data$train.features,
    train.test.data$train.labels, group.lasso.results$w, clustering.results$groups)
```

```

group.scores <- determine.group.scores(train.test.data$train.features, train.test.data$train.labels,
  group.lasso.results$w, clustering.results$groups)

## [1] "Time for determining group scores: 0.02"

group.members <- determine.group.members(train.test.data$train.labels, group.scores,
  group.error.changes, test.classes = 1)$group.members

## [1] "Time for determining group members: 0.00"

save(group.lasso.results, group.error.changes, group.scores, group.members,
  file = sprintf("%s/group_lasso_results.Rdata", res.dir))

```

Finally, motifs can be generated using HOMER by invoking the function `group.motifs`

```

group.motifs(res.dir, dictionary.file, no.cores = 1, org = "hg19", test.classes = 1)

```