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

Manu Setty[1], Christina Leslie

Computational Biology Program, MSKCC,
New York
[1]manu@cbio.mskcc.org

January 18, 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

The R package can be downloaded from `https://bitbucket.org/leslielab/seqgl`. SeqGL was tested using R 3.0.2.

## 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 Pax5 ChIP-seq in GM12878, a lymphoblastoid cell line.

```
peaks.file <- system.file("extdata/gm12878_top_pax5_peaks.bed", package = "SeqGL")
peaks <- read.table(peaks.file, header = TRUE)
head(peaks)
```

```
##   chrom chromStart  chromEnd strand score summit  name
## 1  chr5   77590380  77590682      *  3132    157 Peak1
## 2  chr2   25585594  25585887      *  3122    145 Peak2
## 3  chr1  109655284 109655582      *  3100    145 Peak3
## 4  chr1  150601613 150601900      *  3100    145 Peak4
## 5 chr10    5882276   5882572      *  3100    148 Peak5
## 6 chr10  112116318 112116697      *  3100    193 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_pax5_peaks.bed", package = "SeqGL")
run.seqGL(peaks.file, out.dir = "seqGL.Test/", data.type = "ChIP", org = "hg19")
```

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.

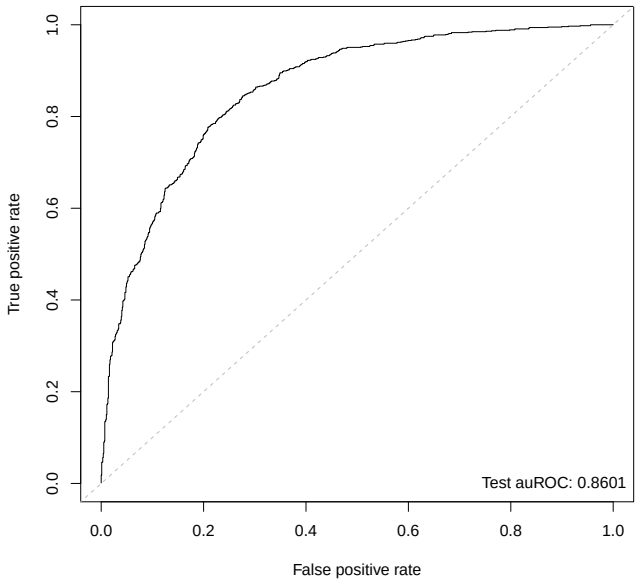


Figure 1: **Test auROC for Pax5 peaks**

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

| Group | Compare Motifs | Test Class | Score | Motif | Motif TF | Denovo Motif Aln |
|---|---|---|---|---|---|---|
| Group13 | Group13 | 1 | 120.39 |  | PAX5(Paired | 0.8681 |
| Group19 | Group19 | 1 | 112.02 |  | MA0067.1_Pax2 | 0.9080 |
| Group11 | Group11 | 1 | 74.84 |  | AP-1(bZIP) | 0.9575 |
| Group3 | Group3 | 1 | 68.20 |  | MA0080.2_SPI1 | 0.8092 |
| Group6 | Group6 | 1 | 47.98 |  | MA0002.2_RUNX1 | 0.8029 |
| Group20 | Group20 | 1 | 27.71 |  | PAX5(Paired | 0.8401 |
| Group4 | Group4 | 1 | 23.41 |  | PU.1-IRF | 0.9227 |
| Group10 | Group10 | 1 | 13.87 |  | PB0164.1_Smad3_2 | 0.6810 |
| Group9 | Group9 | 1 | 10.34 |  | POL010.1_DCE_S_III | 0.7894 |
| Group7 | Group7 | 1 | 8.08 |  | MA0081.1_SPIB | 0.7846 |

Figure 2: **Group scores and motifs for Pax5 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 Pax5 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 recommed 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 matriices 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 organims 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: 0.53 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 : 1.50"
## [1] "Selecting top features..."
## [1] "Time for selecting features: 0.37 minutes"
## [1] "Package and return..."
## [1] "Total time for constructing data: 2.43 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: 2.22 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 ro 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 evaulation 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: 1.48 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: 0.00"

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: 0.02"

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)
```