Analyzing high-throughput data with Clipper

Xinzhou Ge, Yiling Chen

2020-10-05

Contents

Introduction

The most common goal of analyzing high-throughput data is to contrast two conditions to reliably screen "interesting features", where "interesting" means "enriched" or "differential". Enriched features are defined as those that have higher expected measurements under the experimental/treatment condition than the background condition, i.e., the negative control. The detection of such enriched features is called enrichment analysis. For example, common enrichment analyses include calling protein-binding sites in a genome from chromatin immunoprecipitation sequencing (ChIP-seq) data and identifying peptides from mass spectrometry (MS) data. In contrast, differential features are defined as those that have different expected measurements (without measurement errors) between two conditions, and the detection of such differential features is called differential analysis. For example, popular differential analyses include the identification of differentially expressed genes (DEGs) from genome-wide gene expression data (e.g., microarray and RNA sequencing (RNA-seq) data) and differentially chromosomal interaction regions (DIRs) from Hi-C data.

This package provides an easy implementation of a general statistical framework Clipper, an p-value-free FDR control method for analyzing high-throughput biological data. It is applicable to both enrichment analysis and differential analysis. This vignette explains the general use of `Clipper` and then demonstrates its use in typical high-throughput analyses including DEG analysis of RNA-seq data, peak calling from ChIP-Seq data, peptide identification from mass spectrometry data, and DIR analysis from Hi-C data. The details and examples of all these analyses will be further introduced below.

Note: if you use `Clipper` in published research, please cite: For questions regarding the use of `Clipper`, please post to https://github.com/JSB-UCLA/Clipper/issues.

General pipeline

Here we show the most basic applications of Clipper in enrichment and differential analysis. 'Clipper requires a minimum of four inputs:

- `score.exp` the measurements from the experimental (treatment) condition
- `score.back` the measurements from the background (control) condition
- `analysis` the type of analysis, "enrichment" or "differential".
- `FDR` the target FDR threshold(s), set to 0.05 by default

We use `Clipper` to find interesting features by contrasting two measurement matrices, one from the experimental condition and one from the background condition. The inputs of `score.exp` and `score.back` should be numeric matrices. The rows of `score.back` and `score.exp` should match and represent the same feature, and their columns represent replicates (biological samples). For enrichment and differential analysis, set `analysis` to be `"enrichmentl"("e")` or `"differential"("d")` respectively.

We demonstrate the genral pepline of `Clipper` using the following simulated datasets: `exp_e` and `back_e` in example of enrichment analysis; and `exp_d` and `back_d` in example of differential analysis.

```
library(Clipper)
```

Enrichment analysis

For enrichment analysis, we use the simulated data `exp_e` and `back_e` as inputs of `score.back` and `score.exp`. In this dataset, there are 10,000 features and the first 1000 features are interesting. For the target FDR threshold(s), we can then use `Clipper` to perform enrichment analysis:

```
#use three FDR thresholds: 0.01, 0.05, 0.1
exp_e[c(1:3, 1001:1003), ]
#>            [,1]        [,2]      [,3]
#> [1,]  4.9914685  5.26705194  3.945790
#> [2,]  4.8837421  4.65166005  6.889625
#> [3,]  5.0160066  5.00388853  4.830051
#> [4,] -0.3823919  1.70146747 -0.335157
#> [5,] -0.9822577  0.75032253  1.722063
#> [6,] -0.3423613 -0.06262634  1.129383
back_e[c(1:3, 1001:1003), ]
#>            [,1]        [,2]       [,3]
#> [1,] -0.6264538  0.18364332 -0.8356286
#> [2,]  1.5952808  0.32950777 -0.8204684
#> [3,]  0.4874291  0.73832471  0.5757814
#> [4,]  0.7391149  0.38660873  1.2963972
#> [5,] -0.8035584 -1.60262567  0.9332510
#> [6,]  1.8060893 -0.05650363  1.8859113
re1 <- Clipper(score.exp = exp_e, score.back = back_e, analysis = "enrichment",
               FDR = c(0.01, 0.05, 0.1))
names(re1)
#> [1] "contrast.score"       "contrast.score.value" "FDR"
#> [4] "contrast.score.thre"  "discoveries"
```

`Clipper` returns a list and its component `discovery` is a list of indices of identified interesting discoveries corresponding to the FDR threshold(s):

```
#indices of identified interesting genes with the second input FDR threshold (0.05)
re1$discoveries[[2]][1:5]
#> [1] 1 2 3 4 5
```

We can then calculate the resulting false discovery proportion (FDP) and power:

```
#FDP (the first 1000 genes are true positives)
trueid <- 1:1000
sum(!re1$discoveries[[1]] %in% trueid)/length(re1$discoveries[[1]])
#> [1] 0.005149331
#power
sum(re1$discoveries[[1]] %in% trueid)/length(trueid)
#> [1] 0.966
```

Differential analysis

In a differential analysis, the two conditions can be treated equally thus you can assign either condition as background/experimental. We use the simulated data `exp_d` and `back_d` as inputs of `score.back` and `score.exp`. In this dataset, there are 10,000 features and the first 2000 features are interesting. For the target FDR threshold(s), we can then use `Clipper` to perform differential analysis:

```
#use three FDR thresholds: 0.01, 0.05, 0.1
exp_d[c(1:3, 1001:1003), ]
#>          replicates1 replicates2 replicates3
#> gene1             40          43          29
#> gene2             27          46          39
#> gene3             33          27          35
#> gene1001           6           9           4
#> gene1002           2           1           2
#> gene1003           1           3           5
back_d[c(1:3, 1001:1003), ]
#>          replicates1 replicates2 replicates3
#> gene1             19          28          27
#> gene2             29          18          29
#> gene3             17          17          13
#> gene1001          24          28          23
#> gene1002          15          16          18
#> gene1003           8          12           6
re2 <- Clipper(score.exp = exp_d, score.back = back_d, analysis = "differential",
               FDR = c(0.01, 0.05, 0.1))
names(re2)
#> [1] "contrast.score"       "contrast.score.value" "FDR"
#> [4] "contrast.score.thre"  "discoveries"
```

`Clipper` still returns a list and its component `discovery` is a list of indices of identified interesting discoveries corresponding to the FDR threshold(s):

```
#indices of identified interesting genes with the second input FDR threshold (0.05)
re2$discoveries[[2]][1:5]
#> [1] 1 3 5 6 7
```

We can then calculate the resulting false discovery proportion (FDP) and power:

```
#FDP (the first 2000 genes are true positives)
trueid <- 1:2000
sum(!re2$discoveries[[1]] %in% trueid)/length(re2$discoveries[[1]])
#> [1] 0.003556188
#power
sum(re2$discoveries[[1]] %in% trueid)/length(trueid)
#> [1] 0.7005
```

DEG identification

Here we demonstrate how to use `Clipper` for DEG identification. set `analysis` to be `"differential"`. The input of `score.exp` and `score.back` should be gene expression matrices with genes as rows and replicates as columns. It is important to ensure the rows of `score.exp` and `score.back` represent the same set of genes in the same order. `Clipper` requires either `score.exp` or `score.back` to have at least two replicates to perform a differential analysis.

Users are recommended to normalize and log-tranform read count matrices before inputing them to `Clipper` (see examples below). The following example uses `edgeR` to preprocess raw read count matrices as an example. Users can also use their preferred normalization methods.

```
library(edgeR)
dataurl = 'http://www.stat.ucla.edu/~jingyi.li/data/Clipper/'
count1 = readRDS(url(paste0(dataurl, 'simcount1.rds')))
count2 = readRDS(url(paste0(dataurl, 'simcount2.rds')))
############ use edgeR to normalize ############
r1 = ncol(count1)
r2 = ncol(count2)
cond_idx = rep(2, r1 + r2)
cond_idx[1:r1] = 1
dat = cbind(count1, count2)
cond_idx = factor(cond_idx)
y <- DGEList(counts=dat,group=cond_idx)
keep <- filterByExpr(y)  ## optional gene filtering
y <- y[keep,,keep.lib.sizes=FALSE]
y <- calcNormFactors(y)
count_norm = cpm(y)
```

Next we apply `Clipper` to preprocessed gene expression matrices. Log-transformation is recommended because empirical evidence indicates that log-transformed read counts are more likely to satisfy Clipper's assumption for FDR control than read counts without log-transformation.

```
##### apply clipper to log2-transformed matrices with target FDR = 0.05
re_clipper = Clipper(score.exp = log(base = 2, count_norm[,1:r1] + 1),
                     score.back = log(base = 2, count_norm[,-(1:r1)] + 1),
                     FDR = 0.5,
                     analysis = "differential")
head(re_clipper$discoveries[[1]]) ###### identified DEG indices at FDR = 0.05
# 34  72  90 118 125 132
```

DIR identification from Hi-C data

To demonstrate the use of `Clipper` to identify differential chromosomal interaction regions, we start with interaction matrices obtained from Hi-C data. Because Hi-C interaction matrices are often stored as sparse matrices, we need to format these matrices into the right form. Suppose we have two interaction matrices under the experimental condition and the another two under the background condition.

```
exp1 = readRDS(url(paste0(dataurl, 'interactionMatrix_exp1.rds')))
exp2 = readRDS(url(paste0(dataurl, 'interactionMatrix_exp2.rds')))
back1 = readRDS(url(paste0(dataurl, 'interactionMatrix_back1.rds')))
back2 = readRDS(url(paste0(dataurl, 'interactionMatrix_back2.rds')))
head(exp1)
#        V1    V2
# 1:1 0e+00 0e+00  63537
# 1:2 0e+00 1e+06  48305
# 2:2 1e+06 1e+06 366193
# 1:3 0e+00 2e+06   4195
# 2:3 1e+06 2e+06  58452
# 3:3 2e+06 2e+06 344527
```

`exp1` is a interaction matrix of resolution 10^6 base pairs. Each row represents the interaction intensity between one pair of regions: the first integer indicates the starting point of a region, the second integer indicates the

starting point of another region, and the third numeric indicates the read counts between these two regions. Only pairs of regions with a non-zero read count are explicitly coded in this matrix; all other pairs have zero read counts.

`Clipper` treats pairs of chromosome regions as features to call DIRs. Therefore, it is important to make sure to match features when we generate input to `score.exp` and `score.back`.

```
### Check if features match
all(exp1[, 1:2] == exp2[, 1:2])
all(exp2[, 1:2] == back1[, 1:2])
all(back1[, 1:2] == back2[, 1:2])
### features are matched!
```

If not matched, users can use the following code to extract shared features and match them.

```
### generate a replicate
exp2_sim = exp2[sample(1:nrow(exp2), size = 0.9*nrow(exp2),replace = F),]
ls_m = list(exp1, exp2_sim, back1, back2)
features = lapply(ls_m, function(x){
  feat = paste0(x[,1],':', x[,2])
  # rownames(x) = feat
  return(feat)
})
features_shared = Reduce("intersect",features)
ls_m_shared = mapply(function(m,f){
  m[match(features_shared, f),]
}, m = ls_m, f = features, SIMPLIFY = F)
### ls_m_shared is a list that contains interaction matrices with matched features
```

Next we apply `Clipper` by collecting replicates under the experimental condition and the background condition.

```
re <- Clipper(score.exp = log(cbind(exp1[,3], exp2[,3])),
        score.back = log(cbind(back1[,3], back2[,3])),
        FDR = c(0.01, 0.05, 0.1),
        analysis = "differential")
head(exp1[re$discoveries[[2]] ,1:2]) ## identified DIRs
#             V1       V2
# 47:112 4.6e+07 1.11e+08
# 48:112 4.7e+07 1.11e+08
# 49:112 4.8e+07 1.11e+08
# 50:112 4.9e+07 1.11e+08
# 51:112 5.0e+07 1.11e+08
# 52:112 5.1e+07 1.11e+08
```

Peak calling

Here we demonstrate how to apply `Clipper` as an add-on to MACS, a popular peak calling method, for a better FDR control. Similar workflows also apply if users prefer other peak calling method such as HOMER.

Extract input data from existing peak calling software MACS

To implement `Clipper` for peak calling, users need to extract two outputs from MACS: a "*_peaks.narrowPeak" file and two "*_pileup.dbg" files, one for the experimental track and the other for the control track. The "*_peaks.narrowPeak" file contains candidate peaks identified by MACS with its second and third rows

indicating the start and end points of the peaks. The "*_pileup.dbg" files contain the base-pair read coverages for genomic regions of interest. See the following code chunk for a glance of these files. See the end of this section for command line codes we used to generate the "*_peaks.narrowPeak" file and the two "*_pileup.dbg" files. In the following example, we use a synthetic ChIP-seq sequence as the experiemntal sample and a real control sample as background sample. The "*_pileup.dbg" files for these two samples are: `'experimental_treat_pileup.bdg'` and `'control_treat_pileup.bdg'`. The peaks called by MACS by contrasting these two samples are in `'twosample_peaks.narrowPeak'`.

```
experimental <- read.table(url(paste0(dataurl, 'experimental_treat_pileup.bdg')))
head(experimental)
#     V1    V2    V3 V4
# 1 chr1     0  9852  0
# 2 chr1  9852  9913  1
# 3 chr1  9913 10150  2
# 4 chr1 10150 10175  1
# 5 chr1 10175 10211  2
# 6 chr1 10211 10256  1
control <- read.table(url(paste0(dataurl, 'control_treat_pileup.bdg')))
head(control)
#     V1   V2    V3 V4
# 1 chr1    0  9811  0
# 2 chr1 9811  9819  1
# 3 chr1 9819  9896  2
# 4 chr1 9896  9947  3
# 5 chr1 9947  9997  4
# 6 chr1 9997 10106  5
macs2.peak <- read.table(url(paste0(dataurl, 'twosample_peaks.narrowPeak')))
head(macs2.peak)
#     V1      V2      V3               V4  V5 V6       V7       V8       V9  V10
# 1 chr1  904144  905821 twosample_peak_1 155  .  9.98306 18.60009 15.51687  425
# 2 chr1  940135  943865 twosample_peak_2 475  . 21.38286 51.57176 47.52760 1770
# 3 chr1 1058992 1060607 twosample_peak_3 218  . 12.25136 25.12054 21.81555  402
# 4 chr1 1247855 1249009 twosample_peak_4 155  .  9.98306 18.60009 15.51687  278
# 5 chr1 1344798 1345386 twosample_peak_5  63  .  6.46653  9.08008  6.38249  250
# 6 chr1 1430427 1432027 twosample_peak_6 130  . 10.31231 16.01712 13.03906 1322
```

Use Clipper to screen candidate peaks

We use the following codes to generate the input of `score.exp` and `score.back` from the two "*_pileup.dbg" files. The resulting vectors `s1` and `s2` contains the read coverages for each base pair. Then we supply `s1` to `score.exp`, `s2` to `score.back`, and use `Clipper` to perform enrichment analysis to obtain a threshold on per-base-pair contrast scores.

```
# create two vectors of read coverage, one for the experimental track and the other for the background
s1 <- rep(experimental$V4, experimental$V3- experimental$V2 )
s2 <- rep(control$V4, control$V3- control$V2)
s1[(length(s1)+1):length(s2)] <- 0
# use Clipper
re <- Clipper(score.exp = matrix(s1, ncol = 1),
              score.back = matrix(s2, ncol = 1),
              analysis = "enrichment")
# the threshold on contrast scores
re$contrast.score.thre
# 6
```

We then use this threshold to screen the candidate peaks identified by MACS. We compare the median of per-base-pair contrast scores within each candidate peak with the threshold output by `Clipper`. Only peaks whose median per-base-pair contrast score is greater or equal to the threshold are identified as peaks. We don't directly use the discoveries output by `Clipper` becasuse they are base-pair features and do not form biologically meaningful peaks. Instead, we use better-defined peak regions by MACS.

```
### compute the median of per-base-pair contrast scores within each candidate peak
median.peak <- sapply(1:nrow(macs2.peak), function(i){
  # start point of the candidate peak
  start <- macs2.peak$V2[i]+1
  # end point of the candiate peak
  end <- macs2.peak$V3[i]
  # the different contrast score is the difference between the two condiitons
  return(median(s1[start:end] -(s2[start:end])))
})
# peaks identified by Clipper
clipper.peak <- macs2.peak[median.peak >= re$contrast.score.thre,]
```

Generation of MACS output files

We start from two files applicable for MACS peak calling, such as two bam files: `experimental.bam` for experimental condition and `control.bam` for background/negative control condition. We use the following shell scripts to obtain the "∗_peaks.narrowPeak" file and two "∗_pileup.dbg" files. See links for tutorials of MACS.

```
macs2 callpeak -t experimental.bam -c control.bam -f BAM -n twosample -B -q 1 --outdir results

macs2 callpeak -t experimental.bam -f BAM -n exp -B -q 1 --outdir results

macs2 callpeak -t control.bam -f BAM -n back -B -q 1 --outdir results
```

Peptide identification as an add-on of a database search algorithm

The enrichment functionality of `Clipper` also allows it to control the FDR of peptide identification as an add-on to database search engines. Here we demonstrate this function using an example of Mascot, a popular database search method for peptide identification. Similar workflows also apply if users prefer other database search methods such as SEQUEST, Byonic and etc. First we load a peptide-spectrum match (PSM) level sample output from Mascot with the target-decoy search strategy:

```
library(openxlsx)
temp = tempfile(fileext = ".xlsx")
dataURL <- paste0(dataurl, 'Archaea%20Mascot%20Targets%20100_FDR.xlsx')
download.file(dataURL, destfile=temp, mode='wb')
dat = read.xlsx(temp)
dataURL <- paste0(dataurl, 'Archaea%20Mascot%20DECOYS%20100_FDR.xlsx')
download.file(dataURL, destfile=temp, mode='wb')
dat_decoy = read.xlsx(temp)
# str(dat)
```

To apply `Clipper` to control FDR of identified PSMs, we treat spectra as features; given a spetrum, we treat its q-value from the target search as a measurement under the experimental condition and its q-value from the decoy search as a measurement under the background condition. Prior to applying Clipper, we use the following code to format search results:

```
### generate features, psms, and scores from the target search
query = paste( dat[, "Spectrum.File" ], dat[, "ScanNum"], sep = ':')
match = paste( dat[, "Spectrum.File"], dat[, "ScanNum"], dat[,"Sequence"],sep = ':')
score = -log(dat[, "Percolator.q-Value"] + 0.01)

dat = cbind.data.frame(query = query, match = match, score = score, stringsAsFactors = F)

### generate features, psms, and scores from the decoy search
query_decoy = paste(dat_decoy[, "Spectrum.File"], dat_decoy[, "ScanNum"],sep = ':')
match_decoy = paste(dat_decoy[, "Spectrum.File"], dat_decoy[, "ScanNum"],
                    dat_decoy[,"Sequence"],sep = ':')
score_decoy = -log(dat_decoy[, "Percolator.q-Value"] + 0.01)

### get rid of the false decoy matches
query_decoy = query_decoy[!match_decoy %in% match]
match_decoy = match_decoy[!match_decoy %in% match]
score_decoy = score_decoy[!match_decoy %in% match]

### re-order decoy queries and scores to match the target output
score_decoy = score_decoy[match(dat$query, query_decoy)]
match_decoy = match_decoy[match(dat$query, query_decoy)]
query_decoy = query_decoy[match(dat$query, query_decoy)]

### check percentage shared
mean(query_decoy %in% query)
length(score_decoy)
# 0.8422165
```

Our goal is to identify spectra whose q-value from the target search is smaller than that from the decoy search; thus we set `analysis` to be `"enrichment"`. Because a smaller q-value is considered better, we feed −log-transformed q-values into `Clipper`.

```
re = Clipper(score.exp = score,
               score.back = score_decoy,
               FDR = 0.01,
               analysis = 'enrichment')
head(match[re$discoveries[[1]]]) ### identified matches in the form of filename:spectrum scan number: pe
# [1] "am190328_006.raw:1572:KYEESGKPR"    "am190328_006.raw:1720:RADSMISDEKER"
# [3] "am190328_005.raw:1784:MMVEVAK"      "am190328_006.raw:1807:YPESNYMHK"
# [5] "am190328_005.raw:1830:MDMVNYNQK"    "am190328_006.raw:1852:MDMVNYNQK"
```