Cell-type-specific-GSA on test dataset

Presented by - Mehak Chopra (December 20, 2024)

PhD Year III, University of Galway, School of Mathematical and Statistical Sciences, Ireland

Postgraduate fellow - Yale University, School of Medicine

Contact details: m.chopra1@universityofgalway.ie (Primary address) / mehak.chopra@yale.edu

Here we will show a simple workflow for carrying out cell type-specific gene set analysis (CT-GSA). First let's load the CAR-seq package which is designed for carrying out cell type-specific differential expression analysis.

installation of mremaR

```
#library("devtools");
#install_github("osedo/mremaR")
```

load the library

```
library(mremaR)
```

installation of CARseq

```
#devtools::install_github("chongjin/CARseq")
library(CARseq)
```

##Simulation here is shown as we are running the test dataset. No need to do simulations on real data.

Now we will simulate gene expression for samples which are a mixture of six different cell types. This defaults to 1,000 genes in 100 cases and 100 controls. In each cell type 5% of genes are assigned a log-fold-change drawn from a normal distribution with a mean of plus/minus 4 and a standard deviation of 0.2.

```
sims <- CTexpSimulation(g = 1000, n = 100, p = 1, m_lfc = 0, sd_lfc = 1)
sims$bulk.expression[c(1:5),c(1:5)] # bulk expression counts
```

```
##        sample.1 sample.2 sample.3 sample.4 sample.5
## gene.1       85      108      100       70       66
## gene.2      364      206      231      266      243
## gene.3       57       52       44       70       54
## gene.4     1671     1492     2006     1539     2792
## gene.5       59       72       57       56       49
```

```
sims$ols.mixture.estimate[c(1:5),] # cell type proportion estimates
```

```
##          cellType.1 cellType.2 cellType.3 cellType.4 cellType.5 cellType.6
## sample.1 0.04284338 0.04242752 0.14776560  0.1468074  0.2077189  0.4124372
## sample.2 0.03033958 0.02842261 0.04032249  0.1365052  0.1731149  0.5912952
## sample.3 0.01313327 0.03405043 0.02455013  0.1280236  0.2666510  0.5335916
## sample.4 0.08292979 0.09591765 0.07422124  0.1667396  0.1706415  0.4095503
## sample.5 0.02736312 0.05567166 0.02238624  0.2031242  0.3604349  0.3310199
```

```
table(sims$trait) # sample traits
```

```
##
##  0  1
## 50 50
```

These outputs can then be passed to the CAR-seq package. The important results for us are the shrunken_lfc and shrunken_lfcSE matrices

```
# this will take a minute or two to run
res <- run_CARseq(
  count_matrix = sims$bulk.expression,
  cellular_proportions = sims$ols.mixture.estimate,
  groups = sims$trait,
  cores = 10    ##number of cores can be changed
)
```

As we are running on a test data, we are simulating a few gene sets. For original data, use open available datasets or the link provided below:

kegg gene sets data

#kegg.gs <- "https://raw.githubusercontent.com/osedo/GSA-MREMA/main/real_data/kegg.RData"

#load(url(kegg.gs))

Since the gene set mentioned above contains genes in ENTREZ IDs, I recommend converting the gene names from Ensembl IDs to ENTREZ IDs. Alternatively, you can load a gene set dataset that uses Ensembl IDs or another dataset with your preferred ID format using ensembldb or gProfiler R packages.

Let's simulate a few gene sets (no need for the real dataset, just load kegg.gs in place of gs)

```
# a few power gene sets, gs1 enriched for DE genes in cellType1, same for gs2 and so on
# we will give sets of 100 approx 20% DE gene
g <- 1000
gs.power <- lapply(c(1:6), function(x){
  ct_de <- which(abs(sims$simulated.lfc[,x]) > log2(2.5))
  ct_nonde <- c(1:g)[!c(1:g) %in% ct_de]
  ct_de <- paste0("gene.", ct_de)
  ct_nonde <- paste0("gene.", ct_nonde)
  genes_per_set <- 100
  p <- mean(abs(sims$simulated.lfc[,x]) > log2(2.5))
  enrichment <- 3
  c(sample(ct_de, genes_per_set*enrichment*p),
    sample(ct_nonde, (genes_per_set-genes_per_set*enrichment*p)))

})
```

```r
# random gene sets
gs.null <- lapply(c(7:20), function(x){
  paste0("gene.", sample.int(1000, 100))
})

gs <- c(gs.power, gs.null)
names(gs) <- paste0("gs", 1:20)
```

Let’s run mrema() on all cell-types

parametric tests

```r
mrema.res <- lapply(c(1:6), function(ct){
  postdata <- data.frame("Ensembl" = rownames(res$shrunken_lfcSE),
                         "effect" = res$shrunken_lfc[,ct],
                         "variance" = res$shrunken_lfcSE[,ct]^2,
                         "pval" = res$p[,ct])
  mrema_res <- mrema(postdata = postdata, raw.gs = gs, threshold = 2, DF = 4, ncores = 1)
  mrema_res
  })
names(mrema.res) <- colnames(res$p)
saveRDS(mrema.res, file = "samplepara.RDS")
```

non-parametric tests – doesnt make any assumptions about the LFC distribution of a group of genes

```r
full.gsa <- lapply(c(1:6), function(ct){
  data <- data.frame("genes" = rownames(res$shrunken_lfc),
                     "lfc" = res$shrunken_lfc[,ct],
                     "lfcSE" = res$shrunken_lfcSE[,ct])
  gsa.res <- REtest(data, log2(2), gs)
  gsa.res

})
names(full.gsa) <- colnames(res$p)[c(1:6)]
saveRDS(full.gsa, file = "samplenonpara.RDS")
```

We can now launch the shiny app to investigate our results

```r
mremApp()   #parametric tests
REshine()   #non-parametric tests
```

If there are any issues related to mremaR package, please contact - d.oshea20@universityofgalway.ie (Dónal O’Shea - Wrote the package)

Queries can also be sent to m.chopra1@universityofgalway.ie / mehak.chopra@yale.edu