rSEA Practicals

Mitra Ebrahimpoor

This practical explains the use of the `R` package `rSEA` for pathways enrichment analysis where the association of multiple feature-sets with a response variable is of interest.

Preparations

Installing packages

Start by downloading and installing the `rSEA` package. You need to install a package only once. Then, always load it using `library("name_of_pkg")` before running the functions. Here we also install two bioconductor packages `limma` and `edgeR` to calculate the p-values.

```
install.packages("rSEA")

if(!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install(c("limma","edgeR"))

###if you are running an older version of R
#source("http://bioconductor.org/biocLite.R")
#biocLite(c("limma","edgeR"))
```

Loading the Data

For this practical we use the Duchenne muscular dystrophy data of [Pescatori2007]. The data have been provided to you as an .RData file, as a genomics convention the expression values and phenotypic data are stored seperately and combined according to the function to be used. To load the data, you may first have to change to the proper working directory. Use the function `setwd` or the menu (File ... change dir ...). Have a look at the data files and check their dimentions with `dim()` function. The expression data have already been normalized using RMA. Do the dimensions match?

```
#setwd("path_to_ur_dir")

load("dmd_dat.RData")
head(dmd_dat)
dim(dmd_dat)

load("dmd_pData.RData")
head(dmd_pData)
dim(dmd_pData)
```

Calculating raw p-values

Now we need to calculate the feature-wise p-values. For this we will use `limma` package, `limma` requires DGEList objects as input, you can create this using `edgeR`. This is already done for you and saved in "dmd_y.RData" file. You can directly load the file and inspet it by printing it on your console. The other two inputs `limma` require are the "voom" object and "design matrix". The voom object is the output of a variance correction algorithm whithin `limma`. The design matrix is the set of variables to be included in the model. We also define the contract we are interested in, the difference between two groups. The fit object created by

limma can be inspected by `topTable()` function. We will extract the raw p-values and corresponding IDs to use with `rSEA`.

```
####create DGEList or load it
#library(edgeR)
#dmd_y <- DGEList(dmd_dat, group=dmd_pData$dmd, samples = dmd_pData )

load("dmd_y.RData")
dmd_y

library(limma)

design <- model.matrix(~1+group+batch,dmd_pData)
head(design)
group_dif <- makeContrasts(groupC-groupDMD, levels=design)
head(group_dif)

vobj<-voom(dmd_y, design, plot=FALSE)
vfit <- lmFit(vobj, design)
vfit <- contrasts.fit(vfit, contrasts=group_dif)
ebfit <- eBayes(vfit)
topTable(ebfit)

ebfit_res<-as.data.frame(ebfit)
dmdp<-data.frame(id=rownames(ebfit_res),pvals=ebfit_res[,9])
```

Evaluate the set of all features

First load the `rSEA` package. For all functions, you can use a "?func_name" to get help and examples. Try "?setTDP" as an example. Now get an overview of the data you can use `setTDP()` function. If you do not provide a set argument, the set of all features is selected. How do you interpret the resulting values?

```
library(rSEA)
setTDP(pvalue = pvals, featureIDs = id, data=dmdp)
```

We can also do a "self-contained" test to see if there are *any* features associated with the outcome. This can be done using the `setTest()` function. According to the TDP estimates, do you expect the resulting p-value to be significant or not?

```
setTest(pvalue = pvals, featureIDs = id,
        data=dmdp, testype = "selfcontained")
```

Testing GO Pathways

To test pathways, w need annotations to link the probe identifiers to feature symbols, gene ontology terms and other gene information. The standard format for annotation in `Bioconductor` is an annotation package. Creating the list of pathways depends on the IDs in your data and their mapping to GO. For now, you can load the saved GOList that we have prepared for this example dataset. See the appendix to learn how to create such a list on your own. GO consists of three separate ontologies: biological process, molecular function and cellular component. Usually only one of these is of interest. We test only cellular component. First, load the GOList and check a few of the pathways to see the structure.

```
load("GOList.RData")
GOList[2]
```

```
setTest(pvalue = pvals, featureIDs = id,
        data=dmdp, testype = "selfcontained")
```

You can evaluate the pathways individually using the set argument from `setTest()` and `setTDP()` functions. You can either specify the name or pass the number for the pathway of choice. As we saw earlier, use `setTDP()` function to get the TDP estimates and the`setTest()` function to perform a self-contained test. How do you intrepret the resulting vaues?

```
setTest(pvalue = pvals, featureIDs = id,
        data=dmdp, set = GOList[["GO:1902495"]],
        testype = "selfcontained")

setTDP(pvalue = pvals, featureIDs = id,
       data=dmdp, set = GOList[["GO:1902495"]])

setTest(pvalue = pvals, featureIDs = id, data=dmdp,
        set = GOList[[491]], testype = "selfcontained")

setTDP(pvalue = pvals, featureIDs = id,
       data=dmdp, set = GOList[[491]])
```

You can also use the competitive argumet for "testtype", what is the null hypothesis being tested now? Can you modify it?

```
setTest(pvalue = pvals, featureIDs = id, data=dmdp,
        set = GOList[["GO:1902495"]], testype = "competitive")

setTDP(pvalue = pvals, featureIDs = id,
       data=dmdp, set = GOList[["GO:1902495"]])

setTest(pvalue = pvals, featureIDs = id, data=dmdp,
        set = GOList[[4]], testype = "competitive")
setTDP(pvalue = pvals, featureIDs = id,
       data=dmdp, set = GOList[[4]])
```

It is also poosible to test all GO terms simultaneously using the `SEA()` function. This will automatically test self-contained and default competitve null hypotheses. You can always type "?SEA" to get more details about the arguments and output.

```
seaGO<-SEA(pvalue = pvals, featureIDs = id, data=dmdp,
              pathlist = GOList, thresh = 0.1)
dim(seaGO)

head(seaGO)

seaGO_sub<-SEA(pvalue = pvals, featureIDs = id, data=dmdp,
              pathlist = GOList,select=1:50)
dim(seaGO_sub)

head(seaGO_sub)
```

As you see this tables are quite big, it is possible to sort and/format the output using `topSEA()` function. Run the code below to get the 30 GO terms with lawest adjusted p-values of the default competitve test. The second table is sorted by the custom competitive test.

```
seaGO_sort1<-topSEA(seaGO, by=Comp.adjP,
                        descending = FALSE, n = 30)
seaGO_sort1

seaGO_sort2<-topSEA(seaGO, by=Comp.0.1.adjP,
                        descending = FALSE, thresh =0.05, n = 30)
seaGO_sort2
```

You can all the names of GOterms to your output by running the follwing codes. Load the `GO.db` package,and uese a simple matching function to add GOterm names to your table.

```
library(GO.db)
ls("package:GO.db")

GOterms<-toTable(GOTERM)
head(GOterms)
GOterms<-unique(GOterms[,c("go_id","Term")])
head(GOterms)
seaGO$Terms<-GOterms$Term[which(GOterms$go_id %in% seaGO$Name)]

head(seaGO)
```

Testing WikiPathways

As for GO terms, all pathways from wikipathways database can also be evaluated. The WikiPathways are save as "wikiList.RData". The chart and the data are sorted by the default competitive test adjusted p-value.

```
load("wikiList.RData")

seaWiki<-SEA(pvalue = pvals, featureIDs = id,
             data=dmdp, pathlist = wikiList)

dim(seaWiki)
head(seaWiki)

seaWiki_sort<-topSEA(seaWiki, by=Comp.adjP,
                     descending = FALSE, n = 20, cover = 0.2)
seaWiki_sort
```

Testing KEGG Pathways

As for GO terms, all pathways from KEGG database can also be evaluated. The KEGG pathways are save as "kegglist.RData". A custom competitive test is also added to the chart and the data are sorted according to this column.

```
load("KEGGList.RData")

seaKEGG<-SEA(pvalue = pvals, featureIDs = id,
             data=dmdp, pathlist = KEGGList, thresh = 0.2)

dim(seaKEGG)
head(seaKEGG)

seaKEGG_sort<-topSEA(seaKEGG, by=Comp.0.2.adjP,
```

```
                      descending = FALSE, n = 50, cover=0.2)
seaKEGG_sort
```

Testing custom Pathways

We can check-out the largest and the most significant KEGG pathways, and see if the overlapping features are significant.

```
seaKEGG_sort2<-topSEA(seaKEGG, by=Size,
                      descending = TRUE, n = 20)
seaKEGG_sort2

lap<-union(KEGGList[[94]], KEGGList[[10]])
length(lap)

setTDP(pvalue = pvals, featureIDs = id, data=dmdp, set =lap)

setTest(pvalue = pvals, featureIDs = id,
        data=dmdp, set =lap, testype = "selfcontained")

setTest(pvalue = pvals, featureIDs = id,
        data=dmdp, set =lap, testype = "competitive")
```

Appendix : Creating Pathway Lists

The only data you need to create a pathlist is an annotation file to link the probe identifiers to gene symbols, gene ontology terms and other gene information. This object is called a bimap object and can be retrived from different databases and even a local library. Here we present two examples, one is the famouse Gene Ontology database, for which a various range of `bioconductor` tools exist. The other is the wikipathways and the `rwikipathways` package.

GO pathways

One standard format for annotation in `Bioconductor` is an annotation package. Annotation packages are readily available in Bioconductor for most commercial chip types. For custom-made arrays or for less frequently used platforms, it is possible to make your own annotation package using the `AnnBuilder` package. AnnotationDbi package is a key reference for learnign about how to use bimap objects. `AnnotationDbi` is used primarily to create mapping objects that allow easy access from `R` to underlying annotation databases. As such, it acts as the `R` interface for all the standard annotation packages. For more information read the help file of `AnnotationDbi`.

To create the bimap for the DMD data, install the `hgu133a.db` package as the DMD were done on Affymetrix hgu133a chips.

Take the following steps to install the required bioconductor packages. For older versions of R, please refer to the appropriate Bioconductor release.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("hgu133a.db")
```

Use `ls()` function to view the list of objects provided with this db package. You can see that a mapping of hgu133 to GO exists which provides the relevent bimap data. Here we only focus only on cellular component.

```
library(hgu133a.db)
ls("package:hgu133a.db")

gobimap<-toTable(hgu133aGO)
gobimap<-gobimap[gobimap$Ontology=="CC",]
head(gobimap)
```

The bimap is converted to a GOList in the required format as below.(This may take a while!)

```
GOIDs<-unique(gobimap$go_id)
GOList<-lapply(GOIDs,
               function(id)
                   gobimap$probe_id[gobimap$go_id==id])

names(GOList)<-GOIDs
#head(GOList)

#make sure the Ids are unique and the paths are non-empty
GOList<-lapply(GOList, unique)
GOList <- lapply(GOList, function(path) if (all(is.na(path))) character(0) else path)

#save(GOList, file="GOList.RData")
```

KEGG pathways

Th procdure for creating KEGG pathways is the same, just use "hgu133aPATH" instead.

```
keggbimap<-toTable(hgu133aPATH)
head(keggbimap)

keggIDs<-unique(keggbimap$path_id)
KEGGList<-lapply(keggIDs,
                 function(id)
                    keggbimap$probe_id[keggbimap$path_id==id])
names(KEGGList)<-keggIDs
head(KEGGList)

#make sure the Ids are unique and the paths are non-empty
KEGGList<-lapply(KEGGList, unique)
KEGGList <- lapply(KEGGList,
                   function(path)
                       if (all(is.na(path))) character(0) else path)

#save(KEGGList, file="KEGGList.RData")
```

Wikipathways

Install the rwikipathways package from Github.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("rWikiPathways")
```

For `wikipathways`, the procedure is a bit more complicated as there is no direct mapping from hgu133 to

wikipathways. Here we initially create the pathlist by ensemble Ids and they conver these to hgu133 using the mappings from `hgu133a` package. (This may take a while!)

```
#Matching the IDs to create list of wikipathways for metabs
library(rWikiPathways)

homoPathways<-listPathwayIds(organism="Homo sapiens")
wikiList<-lapply(homoPathways,
                 function(x)
                     getXrefList(pathway = x, systemCode="En"))

#converst ensembl_id to hgu133
hgu2ENSEMBL<-toTable(hgu133aENSEMBL)

wikiList<-lapply(wikiList, function(x)
    hgu2ENSEMBL$'probe_id'[which(hgu2ENSEMBL$'ensembl_id' %in% x)])

names(wikiList)<-homoPathways

#make sure the Ids are unique and the paths are non-empty
wikiList<-lapply(wikiList, unique)
wikiList <- lapply(wikiList, function(path) if (all(is.na(path))) character(0) else path)

#save(wikiList, file="wikiList.RData")
```