TD Analyse Différentielle sous R/Bioconductor

Ghislain Bidaut, Plateforme Cibi, CRCM, Aix-Marseille Université

24/03/2022

1 Introduction

Dans ce TD, nous allons faire une **analyse différentielle** sous **R/Bioconductor** basée sur les données que nous avons commencé à analyser à l'étape précédente et pour lesquelles nous avons générées un fichier de comptage.

Ces données ont été générées par Yusenko *et al* et publiées dans "Monensin, a novel potent MYB inhibitor, suppresses proliferation of acute myeloid leukemia and adenoid cystic carcinoma cells". *Cancer Lett* 2020 Jun 1;479:61-70. PMID: 32014461.

L'objectif de l'analyse différentielle que nous allons mettre en oeuvre ici est de **comparer l'expression** de cellules humaines THP-1 en **présence ou absence d'un traitement sous Monensim**.

1.1 Organisation de ce TD:

- Dans une première partie, nous allons voir comment utiliser **edgeR** pour extraire les gènes différentiellement exprimés sur un la base d'une comparaison sur un **facteur unique** (méthode `exactTest`).
- Dans une seconde partie (Sous forme de TP) , nous allons utiliser une seconde méthode (*Generalized Linear Models*) méthode permettant de tenir compte de **plusieurs facteurs** dans la comparaison des conditions expérimentales (méthode `glmQLTest`).

2 Données

Il est possible de retrouver les données relatives à cet expérimentation sur le site de **GEO** sous le numéro d'accession **GSE130657**: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE130657 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE130657)

2.1 Rappel: Tableau récapitulatif des données

SRA	GEO	Treatment	Sample
SRR9005674	GSM3746500	THP0	1
SRR9005675	GSM3746501	THP0	2
SRR9005676	GSM3746502	THP0	3
SRR9005677	GSM3746503	M	1
SRR9005678	GSM3746504	M	2
SRR9005679	GSM3746505	M	3

- **THP0**: Pas de traitement
- **M**: Cellules traités avec Monensin.

2.2 Récupération des données de comptage précédemment générées

Dans la suite, nous allons exploiter le fichier de comptage `counts_GSE130657.txt` généré lors du TD d'analyse RNA-Seq précédent. Nous allons aussi récupérer le fichier `targets_GSE130657.txt`.

Le traitement des données sera fait en **Local**.

Ces fichiers sont disponibles sur

https://mycore.core-cloud.net/index.php/s/pvmUZqNjTpYybCc?path=%2FTD_RNA_Seq (https://mycore.core-cloud.net/index.php/s/pvmUZqNjTpYybCc?path=%2FTD_RNA_Seq)

2.3 Chargement des données dans R

Charger le fichier de comptage dans une variable `expression_raw_counts` et le fichier *targets* dans une variable `targets` à l'aide de `read.table`.

Visualiser ces données dans RStudio (par exemple avec la commande `View`).

Remplacer les en-têtes de colonnes de `expression_raw_counts` par le nom de l'échantillon pour correspondre aux lignes du tableau `targets`.

2.3.1 Solution:

```
# Charger les données
expression_raw_counts <- read.table("counts_GSE130657.txt", sep='\t', header = TRUE, )
targets <- read.table("targets_GSE130657.txt", sep='\t', header = TRUE)


cnames_1 <- gsub("staraligned.", "", colnames(expression_raw_counts))
cnames_2 <- gsub(".trim_Aligned.sortedByCoord.out.bam", "", cnames_1)
colnames(expression_raw_counts) <- cnames_2
```

```
# Visualiser les données

View(targets)
View(expression_raw_counts)
```

2.4 ID Conversion with BiomaRt

Nous disposons d'identifiants de gènes *Ensembl*. Nous souhaitons les adosser à d'autres types d'identifiants:

- Identifiants NCBI **EntrezGene**
- Symboles de gènes (**Gene Symbols**)

Pour cela, nous allons utiliser la base de données BioMart:

Durinck S, Spellman P, Birney E, Huber W (2009). "Mapping identifiers for the integration of genomic datasets with the R/Bioconductor package biomaRt." *Nature Protocols*, 4, 1184–1191.

Il existe une librairie Bioconductor, que nous allons installer et utiliser:

```
library(biomaRt)
```

Avec sa documentation:

https://bioconductor.org/packages/release/bioc/vignettes/biomaRt/inst/doc/accessing_ensembl.html (https://bioconductor.org/packages/release/bioc/vignettes/biomaRt/inst/doc/accessing_ensembl.html)

2.5 Conversion de données avec BioMart

Nous allons nous connecter à la base de données `genes` , puis au jeu de données `hsapiens_gene_ensembl` .

Pour connaitre les base de données et les jeux de données disponibles, nous utilisons les commandes `listEnsembl()` et `listDataset()` .

```
# Liste des bases de données BiomaRt disponibles
# listEnsembl()
ensembl <- useEnsembl(biomart = "genes")

# Liste des jeux de données disponibles
# datasets <- listDatasets(ensembl)
# searchDatasets(mart = ensembl, pattern = "hsapiens")

ensembl <- useDataset("hsapiens_gene_ensembl", useMart("ensembl"))
```

Puis, nous allons utiliser la fonction getBM pour récupérer les attributs `entrezgene_id` et `hgnc_symbol` .

```
ensembl.GID <- expression_raw_counts$Geneid

# Liste des attributs
# attributes <- listAttributes(ensembl)

ensembl.GID.biomart.annot <- getBM(filters= "ensembl_gene_id_version",
                        attributes= c("ensembl_gene_id_version",
                                      "entrezgene_id","hgnc_symbol" ),
                        values=ensembl.GID, mart=ensembl)

annotated_expression_counts <- merge(ensembl.GID.biomart.annot,
                                     expression_raw_counts,
                                     by.x="ensembl_gene_id_version",
                                     by.y="Geneid")
```

3 Analyse de gènes différentiellement exprimés par **edgeR**

3.1 Introduction à edgeR

EdgeR (Robinson *et al* 2010) a été initialement développé pour l'analyse de données **SAGE**, technologie pour laquelle l'expression d'un gène est exprimé de manière *discrète*, par opposition aux microarrays, où les gènes sont exprimés de manière *continue*.

Pour fonctionner, edgeR nécessite **au moins** 1 réplicat dans l'une des conditions expérimentale à comparer.

EdgeR modélise la matrice de comptage (ech i, gène g) suivant le modèle suivant:

$y_{gi} \sim NB(M_i p_{gj}, \phi_g)$

NB: Distribution binomiale négative, M_i taille librairie, ϕ_g dispersion du gène g, p_{gj}, l'abondance relative du gène g dans le groupe expérimental j.

Ce modèle permet de séparer les variations entre les échantillons (ϕ_g) des variations entre conditions expérimentales p_{gj}.

3.2 Fonctionnement de edgeR:

- Un modèle de vraisemblance permet d'estimer la *dispersion* de chaque gène.
- Une procédure bayésienne permets de centrer les dispersions en une valeur consensus. Cette valeur est ensuite intégrée dans le modèle.
- Un test similaire à un test exact de **Fisher** est appliqué sur les données modélisées pour déterminer la significativité de l'expression différentielle des gènes.

3.3 Formatage des données pour traitement par edgeR

D'abord, on charge la librairie `edgeR` :

```
library(edgeR)
```

```
## Le chargement a nécessité le package : limma
```

Nous allons ensuite utiliser la fonction `DGEList` pour créer un objet utilisable par EdgeR.

Effectuer les commandes suivantes:

```
expression_counts <- annotated_expression_counts[,9:ncol(annotated_expression_counts)]

identical(targets$SRA_ID, colnames(expression_counts))
```

```
## [1] TRUE
```

```
expression_annotation <- annotated_expression_counts[,1:8]
```

Puis, visualiser les objets `expression_counts` et `expression_annotation`

```
head(expression_counts)
head(expression_annotation)
```

Puis créer l'objet edgeR proprement dit:

```
y <- DGEList(counts=expression_counts,
             genes=expression_annotation, group = factor(targets$Treatment))
```

3.4 Filtrage des comptages

Il est important de filtrer les gènes trop peu exprimés dans les différents échantillons, car ils peuvent interférer avec les modèles statistiques sous-jacents.

A l'aide de la fonction `filterByExpr` (doc EdgeR), Je demande de garder les gènes:

- Exprimés dans au moins **2** échantillons dans un groupe
- Ayant au moins un compte brut de **15** dans au moins **1** échantillons
- Ayant au moins un compte brut total de **100**
- Ayant un identifiant `entrezgene` et `hgnc_symbol`

3.4.1 Solution:

```
keep <- filterByExpr(y, min.total.count=100, min.prop = 0.6, min.count=15) &
        !is.na(y$genes$entrezgene_id) &
        !is.na(y$genes$hgnc_symbol)

y <- y[keep, , keep.lib.sizes=FALSE]

# Le `keep.lib.sizes=FALSE` force le re-calcul des tailles de librairies.
```

4 Contrôles qualité (QC)

4.1 Tailles de librairies

Faire un plot de la taille des libraires:

```
barplot(y$samples$lib.size, names=colnames(y), las=2)

title("Library sizes")
```

4.2 Distribution des comptages

La fonction `cpm` (Counts per Millions) permets de récupérér les comptages.

Exercice:

- Faire un `boxplot` des logs(CPMs) par échantillons.
- Ajouter au graphe une ligne horizontale matérialisant les médianes des CPMs.

Solution:

```
logcounts <- cpm(y, log=TRUE)
boxplot(logcounts, xlab="", ylab="Log2 counts per million", las=2)
abline(h=median(logcounts), col="blue")
title("Boxplots of logCPMs (unnormalised)")
```

4.3 Plots QC: Distribution des comptages

On peut faire un plot de la distribution des comptages à l'aide de la fonction `plotDensity` de la librairie `affy` .

```
library(affy)

epsilon <- 1.0
col <- unique(palette()[as.factor(targets$Treatment)])

plotDensity(log10(expression_counts + epsilon), lty=1, col, lwd=2)
grid()
legend("topright", legend=unique(targets$Treatment), col, lwd=2, cex = 0.7)
```

Excercice:

Faire une couleur unique par échantillons

4.3.1 Solution

```
# Couleur unique par échantillons
col <- unique(palette()[as.factor(targets$Sample)])
plotDensity(log10(expression_counts + epsilon), lty=1, col)
grid()
legend("topright", legend=unique(targets$Sample), col, lwd=2, cex = 0.7)
```

5 Normalisation

5.1 Nécéssité de la normalisation

EdgeR est conçu principalement pour la détection de gènes différentiellement exprimés, mais pas pour estimer les niveaux d'expression absolus.

Pour cette raison, il n'est **pas nécessaire** de normaliser les données **avant de faire appel aux routines de détection de gènes différentiellement exprimés**.

5.2 Normalisation par edgeR en pratique

Les effets purement techniques devraient être lissés devant la mesure d'expression différentielle. Par exemple, la taille des gènes influence grandement les valeurs de comptage mais cela n'a pas d'importance car cette grandeur est la même pour tous les échantillons.

Une autre grandeur importante qui affecte les valeur de comptage est la profondeur de séquençage de chaque librairie. edgeR en tient compte pour ajuster les valeurs d'expression différentielle de manière automatique.

Il est par contre nécessaire d'ajuster pour les effets **des biais technique importants**, comme les **effets de lots** en amont de l'analyse.

Il existe une autre source technique qui peut influencer l'expression et la détection des gènes différentiellement exprimés. Il faut se rappeler que le RNA-seq ne donne pas l'expression absolue de chaque ARNm dans la cellule mais plutôt l'**abondance relative** de ces ARNm.

Cela peut poser problème si un petit nombre de gènes a une expression très élevée dans certains échantillons et pas dans d'autres. Cela signifie que les gènes vont **"consommer"** des *reads* et faire baisser proportionnellement le nombre de *reads* alloués aux autres gènes. Ce sous-échantilonnage des autres gènes pourrait être interprété comme une fausse expression différentielle.

La méthode **calcNormFactor** du package **edgeR** calcule des valeurs d'expression normalisés en utilisant les TMM (*Trimmed Mean of M-Values*) en **tenant compte de ces biais**.

```
y <- calcNormFactors(y, method = "TMM")
y$samples
```

```
##            group lib.size norm.factors
## SRR9005674  THP0 23705619    0.9780126
## SRR9005675  THP0 22479913    0.9708411
## SRR9005676  THP0 23475241    0.9856558
## SRR9005677     M 22782093    1.0282744
## SRR9005678     M 22222318    1.0257079
## SRR9005679     M 20846555    1.0130931
```

5.3 Plot MDS (Multidimensional Scaling)

```
plotMDS(y)
```

Ajout de labels

```
plotMDS(y, labels = targets$Treatment)
```

6 Estimation de la dispersion

6.1 Pourquoi estimer la dispersion ?

L'estimation de la dispersion pour chaque gène permet de donner une significativité statistique à son expression différentielle en tenant compte de sa variance d'expression (ou dispersion).

edgeR utilise un estimateur de dispersion appelé *quantile-adjusted conditional maximum likelihood* (**qCML**) pour les expérimentations avec un facteur unique.

Pour les designs plus compliqués, il utilise le *Cox-Reid profile-adjusted likelihood* (CR) associé à un modèle généralisé linéaire (*GLM*) pour tenir compte de toutes les sources de variations.

Je propose de travailler d'abord avec **qCML** puis de refaire une analyse avec **CR** et de faire une comparaison du résultat.

```
y <- estimateDisp(y)
```

```
## Using classic mode.
```

6.2 Visualisation de la dispersion par PlotBCV

PlotBCV = Plot Biological Coefficient of Variation

```
plotBCV(y)
```

6.3 Test pour les gènes significativement différentiellement exprimés

Une fois la dispersion calculée, on peut ensuite appliquer le test exact par la fonction `exactTest` . Ce test vérifie l'hypothèse nulle sur l'expression différentielle des gènes sur les deux conditions comparées en utilisant une distribution binomiale négative.

```
et <- exactTest(y)
```

6.4 Extraction des gènes par topTags

La fonction `topTags` permet d'extraire le résultat du test pour chaque gène, en y appliquant des filtres (test multiples, tri, n).

Exercice:

On sort les 1000 gènes les plus significativement exprimés par la fonction `topTags` (p-valeurs ajustées les plus significatives). Ensuite, on sauve le résultat dans un fichier `top.diff.simple.test.txt` au format **texte délimité par des tabulations**.

6.4.1 Solution

```
tt <- topTags(et, adjust.method = "BH", sort.by = c("PValue"), n=1000)
colnames(tt)
```

```
##  [1] "ensembl_gene_id_version" "entrezgene_id"
##  [3] "hgnc_symbol"             "Chr"
##  [5] "Start"                   "End"
##  [7] "Strand"                  "Length"
##  [9] "logFC"                   "logCPM"
## [11] "PValue"                  "FDR"
```

```
write.table(tt, file = "top.diff.simple.test.txt", sep = "\t",
            row.names = F, quote = FALSE)
```

7 TP: Analyse avec le modèle *Cox-Reid* d'estimation de la vraisemblance.

7.1 Enoncé

Dans cette analyse, nous allons utiliser l'estimateur *Cox-Reid profile-adjusted likelihood (CR)* associé à un modèle linéaire généralisé.

1. Faire une matrice de *design* incluant l'information d'échantillon. Calculer la dispersion en donnant l'argument `design` à `estimateDisp` .
2. Faire un test statistique permettant d'extraire les gènes différentiellement exprimés en utilisant les fonctions `glmQLFit` et `glmQLTest` à la place de la fonction `exactTest` .
3. Extraire la liste des gènes avec `topTags` et la sauver dans un fichier.

7.2 Element de solution: Matrice de design

Pour comparer les échantillons entre eux en fonction des variables expérimentales étudiées, il faut créer une matrice de *design*, qui permet de formaliser l'information liée à chaque échantillon de manière **binaire**.

Par exemple, voici une matrice de design simple pour faire une comparaison entre groupes (`Treatment`).

Le terme `~0` représente l'intersection (*intercept*).

```
Treatment <- factor(targets$Treatment)
design <- model.matrix(~0+Treatment)
colnames(design) <- c("THP0", "THP0.vs.M")

design
```

```
##   THP0 THP0.vs.M
## 1    0         1
## 2    0         1
## 3    0         1
## 4    1         0
## 5    1         0
## 6    1         0
## attr(,"assign")
## [1] 1 1
## attr(,"contrasts")
## attr(,"contrasts")$Treatment
## [1] "contr.treatment"
```

7.3 Solution

```
# Créer une matrice de design en **spécifiant les échantillons dans l'intersection** (t
erme `~Sample` dans l'appel à `model.matrix`).


Sample <- factor(targets$Sample)
Treatment <- factor(targets$Treatment)

design <- model.matrix(~Sample+Treatment)
colnames(design) <- gsub("group", "", colnames(design))

rownames(design) <- colnames(y)


# Nous utilisons un modèle basé sur la quasi-likelihood (QL), plus robuste en cas de de
sign multi-factoriel (Fonction `glmQLFit`).

# On ajoute l'argument design dans le calcul de dispersion.

y <- estimateDisp(y, design = design)
fit <- glmQLFit(y, design)


# On peut donc ensuite appliquer le test de détection des gènes différentiellement expr
imés.

qlf <- glmQLFTest(fit)


# On extrait les gènes différentiellement exprimés avec topTags


tt <- topTags(qlf, adjust.method = "BH", sort.by = c("PValue"), n=1000)
colnames(tt)
write.table(tt, file = "top.diff.qlf.txt", sep = "\t",
            row.names = F, quote = FALSE)
```

7.4 Sauver les objets dans un fichier

On sauve les objets `y`, `et`, `targets`, `annotated_expression_counts` dans un fichier `Rdata`.

```
save(y, et, targets, annotated_expression_counts, file="diffAnalysis.RData")
```