Транскриптомика: RNASeq. Анализ данных с использованием R/BioConductor

На прошлом занятии мы познакомились со средой R и попробовали свои силы в анализе данных биочипов (microarray). В этом разделе мы будем изучать подходы к анализу RNASeq-данных.

Получение данных

Мы будем использовать данные RNASeq-эксперимента содержащего экспрессию генов для трех образцов сквамозно-клеточной карциномы и парных образцов здоровой ткани. Целью анализа будет определение дифференциально экспрессирующихся генов. Исходный файл взят из работы Tuch et al,

http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0009317

Этот пример содержится в пакете sSeq из Bioconductor. Загрузим данные:

```
source("https://bioconductor.org/biocLite.R")
biocLite("sSeq")

## package 'sSeq' successfully unpacked and MD5 sums checked
...

library(sSeq)
data(Tuch)
```

Данные загрузились в dataframe, который называется countsTable. Для работы с данными RNASeq мы будем использовать пакет edgeR. Проверьте, установлен ли данный пакет в системе, используя функцию library(). Если нет – установите пакет через Bioconductor по ссылке:

http://www.bioconductor.org/packages/release/bioc/html/edgeR.html,

аналогично тому как мы это делали на предыдущем занятии. После установки не забудьте подключить пакет:

```
library(edgeR)
```

Сконвертируем наши данные в объект класса DGEList, поскольку пакет edgeR работает именно с таким типом данных

```
y <-DGEList(counts=countsTable[,1:6], genes=rownames(countsTable))
```

Обратите внимание, что колонки rawdata с 1 по 6 – это количество ридов, имена генов помещаем в отдельную колонку из названий строк. Посмотрим на внутреннюю структуру полученного объекта y:

```
y
```

```
## Anobjectofclass "DGEList"
## $counts
##              N8   N33   N51  T8  T33  T51
## NM_000014  2242  2285 15121 261  597 1991
## NM_144670 11731 13308  6944 912 3071 1160
## NM_017436   162   111   751 296  362  182
## NM_015665   199   215   512  81  344  342
## NM_023928   470   573   690 710 1112  728
## 10448 morerows ...
##
## $samples
##     group lib.size norm.factors
## N8      1  7742608            1
## N33     1 15581382            1
## N51     1 20933491            1
## T8      1  7126839            1
## T33     1 13842297            1
## T51     1 14760103            1
##
## $genes
##                genes
## NM_000014 NM_000014
## NM_144670 NM_144670
## NM_017436 NM_017436
## NM_015665 NM_015665
## NM_023928 NM_023928
## 10448 more rows ...
```

Как видно, у объекта есть поля counts, samples и genes.

Фильтрация и нормализация

Сперва уберем гены, которые не имеют Entrez GeneID:

```
library(org.Hs.eg.db)
idfound <-y$genes$genes %in%mappedRkeys(org.Hs.egREFSEQ)
y <-y[idfound,]
dim(y)
```

```
## [1] 10431     6
```

Для того, чтобы узнать сколько переменных(генов) осталось после такой фильтрации мы использовали функцию dim() которая возвращает количество строк и столбцов матрицы, также можно просто ввести название переменной как команду.

Добавим к нашей аннотации генов колонку с Entrez GeneID. В качестве ключа будем использовать genes, Получим словарь соответсвий genes – EntrezGeneID из пакета org.Hs.eg.db

```
egREFSEQ<-toTable(org.Hs.egREFSEQ)
```

```
head(egREFSEQ)
```

```
##   gene_id    accession
## 1       1    NM_130786
## 2       1    NP_570602
## 3       2    NM_000014
## 4       2 NM_001347423
## 5       2 NM_001347424
## 6       2 NM_001347425
```

Часто (как и в нашем случае) одному EntrezGeneID соответствуют несколько RefseqID Далее отмаппируем RefSeq из переменной y по колонке accession в таблице egREFSEQ, т.е. найдем индексы вхождения первого во второе функцией match()

```
m <-match(y$genes$genes, egREFSEQ$accession)
```

Добавим колонку с названием EntrezGene в таблицу y$genes

```
y$genes$EntrezGene <-egREFSEQ$gene_id[m]
head(y$genes)
```

```
##                genes EntrezGene
## NM_000014 NM_000014          2
## NM_144670 NM_144670     144568
## NM_017436 NM_017436      53947
## NM_015665 NM_015665       8086
## NM_023928 NM_023928      65985
## NM_024666 NM_024666      79719
```

Перейдем от уровня транскриптов к генам. Идеологически воспользуемся тем же приемом, который мы использовали на прошлом занятии – среди нескольких транскриптов соответствующих одному гену выберем транскрипт с наибольшим количеством отмаппировавшихся ридов.

```
o <-order(rowSums(y$counts), decreasing=TRUE)
```

Переменная o – это индексы транскриптов с наибольшей суммой по строкам (т.е. наибольшее количество ридов во всех образцах), отсортированные по убыванию. Перемешаем порядок следования транскриптов в переменной y согласно переменной o

```
y<-y[o,]
```

Обращаем внимание, что на этом этапе только изменился порядок следования генов, но не их число. Далее получим индексы строк с дуплицированным (неуникальным) названием гена и удалим такие строки (отрицание обозначается символом ‘!’). Напоминаем, что транскрипты у нас отсортированы по убыванию суммарного количества ридов.

```
d <-duplicated(y$genes$EntrezGene)
y <-y[!d,]
```

Сколько генов осталось после такой фильтрации?

Поскольку мы отбрасывали транскрипты, пересчитаем общий размер библиотеки в каждом образце (т.е. сумммарное кол-во ридов):

```
y$samples$lib.size <-colSums(y$counts)
```

Затем проведем нормализацию:

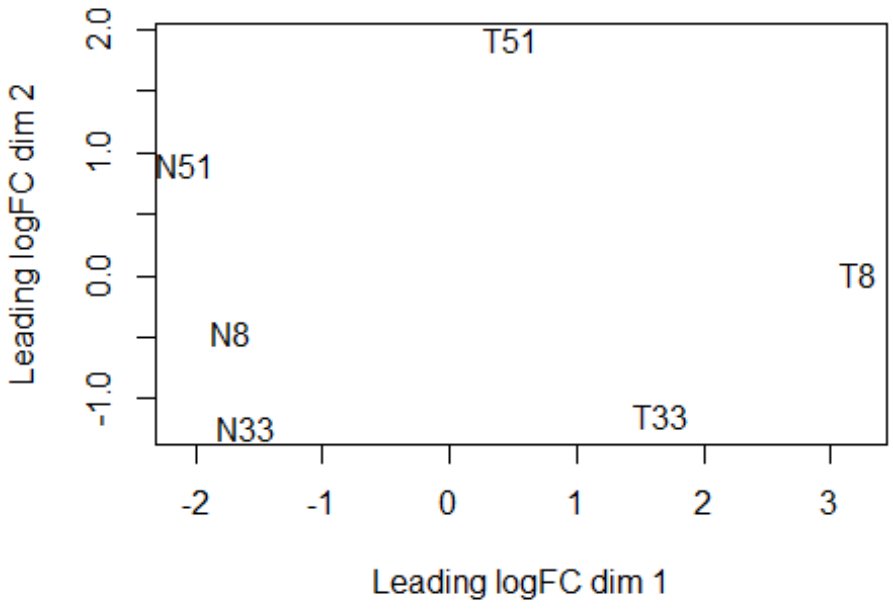
```
y <-calcNormFactors(y)
y$samples
```

```
##     group lib.size norm.factors
## N8      1  7731838    1.1532951
## N33     1 15541495    0.6635890
## N51     1 20896017    1.0360055
## T8      1  7116752    1.0920680
## T33     1 13798153    0.9646714
## T51     1 14732415    1.1972069
```

Разведочный анализ

Первым шагом анализа является проверка образцов на наличие выбросов и общее понимание взаимозависимостей между данными. Полезным инструментом здесь являются техники отображения многомерных данных на плоскости – многомерное шкалирование или анализ главных компонент. Функция plotMDS() создает график, где расстояния между точками приблизительно соответствуют биологическому коэффициенту вариации

```
plotMDS(y)
```



Обратите внимание, что опухолевые образцы (с суффиксом T, tumor) имеют более высокую дисперсию по сравнению с нормальной тканью (суффикс N, normal)

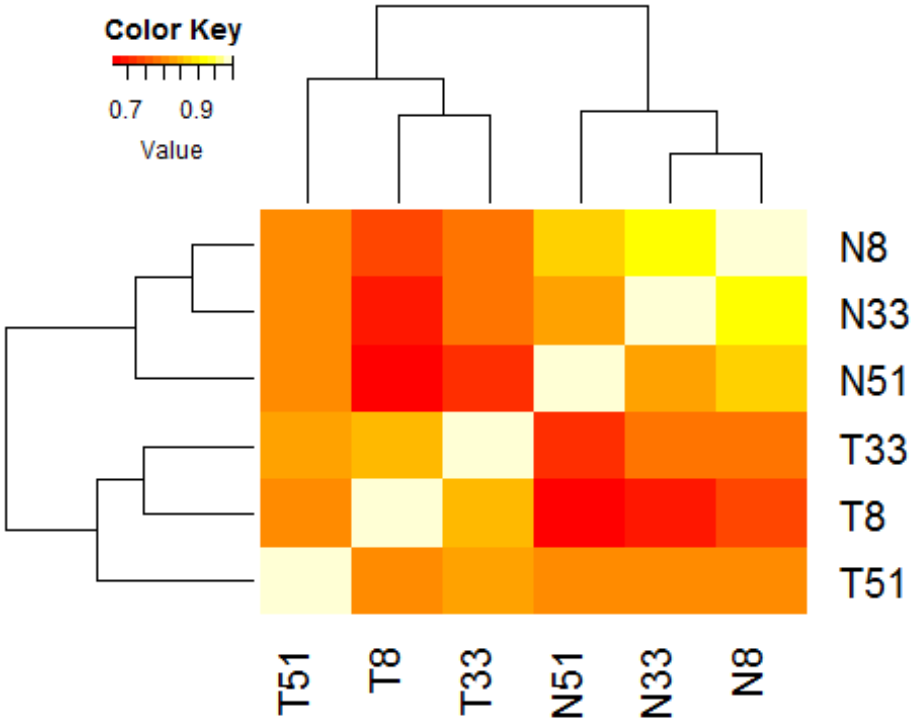
Другим полезным инструментом является вычисление матрицы попарных корреляций между образцами. Для расчета корреляции используется функция cor(). Обратите внимание на аргумент method, он используется для задания метода расчета коэффициента корреляции, например непараметрической корреляции по Спирмену.

```
cor.mtx =cor(y$counts, method ="spearman")
cor.mtx
```

```
##            N8       N33       N51        T8       T33       T51
## N8  1.0000000 0.9159547 0.8746930 0.7319424 0.7839202 0.8113873
## N33 0.9159547 1.0000000 0.8302185 0.6942722 0.7773319 0.7989853
## N51 0.8746930 0.8302185 1.0000000 0.6585028 0.7120626 0.8042973
## T8  0.7319424 0.6942722 0.6585028 1.0000000 0.8518310 0.8040026
## T33 0.7839202 0.7773319 0.7120626 0.8518310 1.0000000 0.8360969
## T51 0.8113873 0.7989853 0.8042973 0.8040026 0.8360969 1.0000000
```

Визуализируем матрицу в виде тепловой карты (heatmap). Как и на прошлом занятии используем функцию heatmap.2() из пакета gplots

```
library(gplots)

heatmap.2(cor.mtx,trace="none",density.info="none")
```



Как происходит группирование образцов – по типу ткани или по парам? Является ли это биологически осмысленным?

Попробуйте использовать менее устойчивый к выбросам линейный коэффициент корреляции по Пирсону. Как изменится кластеризация образцов? Подсказка: прочитайте справку к функции cor() и поменяйте соответствующим образом параметр method.

Дифференциальная экспрессия

На основе корреляционной матрицы сформируем аннотации для образцов: вектора обозначающие попарную зависимость образцов (Patient) и тип ткани (Tissue)

```
Patient <-factor(c(8,33,51,8,33,51))
Patient

## [1] 8  33 51 8  33 51
## Levels: 8 33 51

Tissue <-factor(c("N","N","N","T","T","T"))
Tissue

## [1] N N N T T T
## Levels: N T
```

Для наглядности объединим все аннотации образцов воедино:

```
anno<-data.frame(Sample=colnames(y),Patient,Tissue)
```

```
anno
```

```
##   Sample Patient Tissue
## 1     N8       8      N
## 2    N33      33      N
## 3    N51      51      N
## 4     T8       8      T
## 5    T33      33      T
## 6    T51      51      T
```

Обратите внимание, что Patient и Tissue объявляются как переменные класса factor, т.е. могут принимать только определенные значения, levels. Далее сформируем т.н. матрицу планирования эксперимента (design matrix), показывающую какое воздействие было применено к каждому образцу. Для этого используем функцию model.matrix()

```
design<-model.matrix(~Patient+Tissue)
rownames(design) <-colnames(y)
design
```

```
##     (Intercept) Patient33 Patient51 TissueT
## N8            1         0         0       0
## N33           1         1         0       0
## N51           1         0         1       0
## T8            1         0         0       1
## T33           1         1         0       1
## T51           1         0         1       1
## attr(,"assign")
## [1] 0 1 1 2
## attr(,"contrasts")
## attr(,"contrasts")$Patient
## [1] "contr.treatment"
##
## attr(,"contrasts")$Tissue
## [1] "contr.treatment"
```

В данном случае у нас три фактора: принадлежность к образцу 33, принадлежность к образцу 51 и принадлежность к опухолевой ткани. Остальные варианты (принадлежность к образцу 8 и принадлежность к нормальной ткани) не задаются, поскольку являются отрицанием одного из уже имеющихся факторов (т.е. эти параметры являются избыточными).

Оценим общий параметр дисперсии для негативного биномиального распределения:

```
y <-estimateGLMCommonDisp(y, design, verbose=TRUE)
```

```
## Disp = 0.16027 , BCV = 0.4003
```

Оценим дисперсию для каждого гена в соответствии с дизайном эксперимента

```
y =estimateGLMTrendedDisp(y, design)
y =estimateGLMTagwiseDisp(y, design)
```

Далее подберем общую линейную модель для каждого гена и проведем тесты на правдоподобность

```
fit =glmFit(y, design)
lrt =glmLRT(fit)
topTags(lrt)

## Coefficient:  TissueT
##                      genes EntrezGene     logFC   logCPM       LR
## NM_001039585 NM_001039585       5737 -5.183371 4.778880 93.17632
## NM_014440       NM_014440      27179 -6.132586 5.440005 91.70450
## NM_005609       NM_005609       5837 -5.472427 6.024843 86.63406
## NM_004320       NM_004320        487 -4.618413 6.002083 79.40469
## NM_001111283 NM_001111283       3479 -3.988142 5.752951 79.20424
## NM_198965       NM_198965       5744  3.917717 5.869731 77.77346
## NM_004533       NM_004533       4606 -5.458166 6.535456 77.47623
## NM_033641       NM_033641       1288  3.656847 5.755176 71.02023
## NM_007168       NM_007168      10351 -3.983246 4.975004 70.47041
## NM_182502       NM_182502     132724 -7.445141 7.678151 69.97678
##                     PValue          FDR
## NM_001039585 4.783701e-22 4.988922e-18
## NM_014440    1.006360e-21 5.247662e-18
## NM_005609    1.305789e-20 4.539358e-17
## NM_004320    5.060574e-19 1.168250e-15
## NM_001111283 5.600969e-19 1.168250e-15
## NM_198965    1.155627e-18 2.001314e-15
## NM_004533    1.343292e-18 2.001314e-15
## NM_033641    3.535810e-17 4.609371e-14
## NM_007168    4.672228e-17 5.414074e-14
## NM_182502    6.000683e-17 6.258113e-14
```

Результат тестирования, переменная lrt, является объектом класса DGELRT и для доступа к ней необходимо использовать специфические функции, например topTags(), которая по умолчанию возвращает первые 10 дифференциально экспрессированных генов согласно столбцу PValue. Для каждого дифференциально-экспрессированного гена приводится logratio (logFC), логарифм соотношения правдоподобия (LR), уровень значимости (p-value), скорректированный уровень значимости (FDR). Для того чтобы получить все дифференциально экспрессированные гены воспользуйтесь функцией decideTestsDGE(), которая для каждого гена возвращает +1,–1 или 0 в зависимости от того, является ли ген достоверно гипер-экспрессированным, гипо-экспрессированным или ген не изменил свою экспрессию. По умолчанию функция использует поправку на множественные сравнения и порог 0.05:

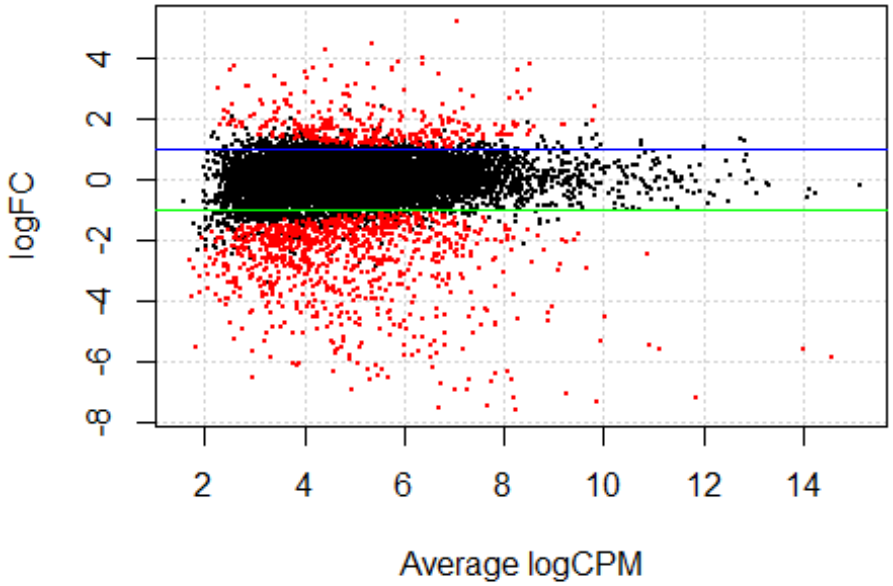
```
de =decideTestsDGE(lrt)
table(de)

## de
##   -1    0    1
##  932 9168  329
```

Как видим, большинство генов (9168) не поменяло достоверно свою экспрессию. Нарисуем зависимость между степенью экспрессии гена (logCPM, counts per million) и изменением его экспрессии (logFC, logarithm fold-change, что эквивалентно logratio). Фактически это означает зависимость между средним значением и дисперсией. Сперва получим названия генов, которые дифференциально экспрессированы:

```
detags =rownames(y)[as.logical(de)]
```

Функция as.logical() вернет значение TRUE (истина) для ненулевых значений (т.е. для -1 и +1) и FALSE (ложь) для нулевых значений. Т.е. тем самым мы получим названия дифференциально экспрессированных генов. Далее используем функцию plotSmear() для построения искомого графика. Указание параметра de.tags подсветит дифференциально экспрессированные гены красным. Добавим на график две синие горизонтальные линии с координатами +1 и -1. Какому изменению экспрессии соответствуют эти значения (при вычислении logFC используется логарифм по основанию два)?

```
plotSmear(lrt, de.tags=detags)
abline(h=c(1), col="blue")
abline(h=c(-1), col="green")
```



К сожалению, многие пакеты R обладающие значительной функциональностью (такой как анализ RNASeq–экспериментов) требуют знания большого количества собственных специфических функций и классов. Так, подобный анализ в пакете DESeq (который также предназначен для работы с RNASeq-данными) потребует тщательного изучения документации и применения собственных функций, отличных от пакета edgeR.