Systems Genetics: Multi-Omics for rare disease

Christian Mertes, Vicente Yepez

10 July 2018 (Room 01.11.018)

Contents

1 Requirements

1.1 R packages

- GenomicRanges
- VariantAnnotation
- snpStats
- ensemblVEP
- gplots
- parallel
- devtools
- data.table
- magrittr
- dplyr
- OUTRIDER (GitHub: https://github.com/gagneurlab/OUTRIDER)

```
source("https://bioconductor.org/biocLite.R")
biocLite(c('GenomicRanges', 'VariantAnnotation', 'ensemblVEP',
        'snpStats', 'gplots', 'devtools', 'data.table',
        'magrittr', 'dplyr'))
devtools::install_github('gagneurlab/OUTRIDER', ref='RELEASE_3_4')
```

1.2 Data

- VCF file (subsetted 1000 genomes)
- Kremer et al. expression count matrix

2 Detection of causal variants

2.1 Basic concepts of rare diseases

1. What is penetrance? In a rare disease cohort with 1000 individuals, 200 were found to contain a variant leading to a disease, but only 80 showed the symptoms. What is the penetrance of the disease?

2.2 Reading and filtering variants

Variant information are stored in the Variant Call Format (VCF). They contain a header and rows with information about a position in the genome with fixed fields like position, reference allele, alternative base, quality score, etc.

1. Read the VCF file using the following code. It contains a **subset** of variants of different samples, with their respective annotation, obtained from the 1000 Genomes Project. How many samples and variants does it contain?

```
library(VariantAnnotation)
library(ensemblVEP)
library(data.table)
library(magrittr)
library(dplyr)
vcf_file <- "./data/vep_anno.vcf.gz"
vcf <- readVcf(vcf_file)
# Samples
colData(vcf)
# Variants
rowData(vcf)
```

2. Check the quality and filter for low quality variants if needed (QUAL $<$ 99). Hint: `?VCF-class`.

3. The variants were annotated using Variant Effect Predictor (VEP). We can extract the full VEP annotation from the vcf object using the following code.

```
# Read the VCF as a Genomic Ranges object
csq <- parseCSQToGRanges(vcf, VCFRowID=rownames(vcf))
length(csq)
# We see more variants than before because
# they are associated to different transcripts of the same gene
```

4. Explore the csq object. How many genes have annotated variants? In which genomic region (biotype) are the variants located?

The function `table()` is useful to count the number of ocurrences of a certain attribute, for example:

```
## Number of times each gene appears; which corresponds to the number of variants per gene
table(csq$SYMBOL)
## Number of annotations per variant
barplot(table(table(names(csq))))
```

5. What is the median annotations per variant? Check variants rs10882655 and rs114713720, what's the difference between them?

2.3 Finding the needle (causing variant) in the hay stack (genome)

2.3.1 Annotation based filtering

Samples `HG00096` and `HG00097` are suspected to have a mitochondrial rare disease. Can you find their variants?

To achive this we need to set up a filtering pipeline considering impact, rarity, SIFT and Polyphen scores. You could look at much more!

1. Filter for High or Moderate impact variant consequences. Hint: *?subset(obj, statement)*
2. Filter for rare variants. Note that if the allele frequency is `NA`, it means the variant is not present in any public database.
3. Filter according to SIFT and PolyPhen scores. Remember they are only applicable to missense variants. Hint: *?grepl(pattern, columnname)*

2.3.2 Genotype based filtering

Now that we have filtered the variants by annotation only, let's have a look at the variants in the samples. The following code accesses the variants matrix and subsets to include only the filtered ones.

```
vars <- unique(names(csq_filter))   # Filtered variants
mat <- geno(vcf[vars])$GT     # function to access the variants matrix
dim(mat)
```

Due to the big amount of samples (now ~2500 individuals) in the 1000 Genome project, they were able through phasing to determine if close heterozygous variants are on the same allele or not. The variant matrix contains either `0|0` (no variant), `1|0` (heterozygous variant on the first allele), `0|1` (het. variant on the second allele) and `1|1` (homozygous variant). Note that the matrix is quite sparse - most fields are `0|0`. Melting is a convenient way to visualize the variants in a tidy way and then remove fields with no variants. Use the following code:

```
vt <- melt(mat) %>% as.data.table()  # mat is the variants matrix from the previous question
setnames(vt, c("Var1", "Var2"), new = c("VariantID", "Sample"))
```

4. Remove values without variants.
5. Merge the variant-sample matrix with the GenomicRanges object to have gene information. Use the following code to convert the Genomic Ranges object to a `data.table`. Hint: *?merge.data.table()* Do you find any heterozygous or compound homozygous variants?

```
csq_dt <- as.data.table(csq_filter)
csq_dt[, VariantID := names(csq_filter)]
csq_dt = unique(csq_dt[,.(SYMBOL, VariantID)])
```

6. Do you find rare homozygous or compound heterozygous variants in samples HG00096 and HG00097? Do you find those variants in other individuals? Make biological sense of those variants.

3 Detection of expression outliers

3.1 Basics for expression data

Load all needed packages

```
library(parallel)
library(stats)
library(gplots)
library(OUTRIDER)
```

Load the expression count matrix. You can find the count matrix here: https://media.nature.com/original/nature-assets/ncomms/2017/170612/ncomms15824/extref/ncomms15824-s1.txt.

```
# Read GTEx count matrix
url <- paste0("https://media.nature.com/original/nature-assets/",
        "ncomms/2017/170612/ncomms15824/extref/ncomms15824-s1.txt")
cts <- read.delim(url)
```

1. How many genes and samples are there? Are all genes expressed? What is a good transformation for count data?
2. Filter out not expressed genes. A naive way of doing it is based on a cutoff of observed reads (`rowMeans(cts)>1`). How many genes are filtered out?

```
# each gene has to have at least one read mean
passedFilter <- rowMeans(cts) > 1
cts <- cts[passedFilter,]
dim(cts)
```

3.2 The negative binomial distribution for count data

1. RNA expression data (count data) is often modeled with a negative binomial distribution. What parametrization does the NB distribution has? What is the meaning of it?
2. Plot densities of the NB with various size parameters and a mean of 30. Hint: *?dnbinom*.
3. Most of the optimization methods use the log-likelihood of a distribution to find the optimal parameters. Give the log-likelihood for the negative binomial distribution.
4. We can use the `optim` function to fit our NB model. `optim` minimizes the problem instead of maximizing it. Therefore we need the negative log-likelihood to fit the parameters of the distribution $NB(k|\mu, \theta)$. We can use the `dnbinom` again for the log-likelihood. Please fill in the code for the loglikelihood. What could be a good initial value for the first parameter?

```
nloglikelihood <- function(par, counts){
    mu <- par[1]
    size <- par[2]

    ...
```

```
}

# fit nb model with optim
fitNB <- function(cts, gene){
    cts4fit <- as.integer(cts[gene,])
    initPar <- c(mu= ..., size=1)
    optim(par=initPar, fn=nloglikelihood, counts=cts4fit)
}
```

5. Now apply it to the gene `ALDH18A1`. What do we get?

3.2.1 Calculation of the one-sided P-value

We have the fit now for a given gene. Now calculate the one-sided P-value. Please fill in the function. Apply it again for the gene 'ALDH18A1'. What do we get?

```
oneSidedPVal <- function(cts, gene, fit){
    cts4predict <- as.integer(cts[gene,])
    mu <- fit$par['mu']
    size <- fit$par['size']

    ...
}
```

4. Lets apply it to all genes (~13k). We will use *mclapply(x, FUN, params, mc.cores=4)* to parallelize the calculation. Here we will use 4 cores in parallel. You can change this according to your hardware. This can take around 30-60 seconds. If you do not have mclapply on your R-version you can use `lapply` instead without the `mc.cores` parameter.

```
# Fit the model per gene
fits <- mclapply(1:nrow(cts), fitNB, cts=cts, mc.cores=4)

# calculate the pvalues per gene based on the fit
pvals <- mclapply(1:nrow(cts), mc.cores=4, function(x){
    res <- oneSidedPVal(cts, x, fits[[x]])
    names(res) <- colnames(cts)
    res
})
# name the results for better handling
names(pvals) <- rownames(cts)
```

5. How do we find out if our model fits our data? What is the function below plotting? Apply this function our gene of interest `ALDH18A1`?

```
# P-values should be uniformly distributed.
# Means we plot the expected P-value against the sorted observed ones.
plotFUN <- function(pvals, gene){
    obsp <- pvals[[gene]]
    expp <- (1:length(obsp)-0.5)/length(obsp)
    plot(-log10(expp),-log10(sort(obsp)), pch=16);
```

```
        abline(0,1, col='red');
        grid()
    }
```

3.3 Check the results and diagnose patients!

1. What is the gene with the lowest P-value for sample (`MUC1404` and `MUC1365` which are equivalent to the VCF samples `HG00096` and `HG00097`)? And is it still significant after multiple correction? Hint to extract data from lists: *sapply(list, "[[", element_name)* and for multiple correction *?p.adjust()*.

Usually the correction is done on the test it self. Here we do the multiple correction per sample. Means for one sample across different tests. Since genes can be correlated we use the `Benjamini-Yekuteli` correction *method='BY'*.

2. A very convenient way to visualize expression outliers per sample is the Volcano plot. It plots the expression based Z-score against the -log10(P-value). The Z-score is defined as $z = \frac{x-\mu}{\sigma}$. How does the volcano plot look like for the patient "MUC1365"? Hint to do row-wise calculations: *apply(mat, 1, sd)*

3.4 Parametrization of the NB distribution

In order to fit the autoencoder we need the gradient of the negative log-likelihood.

1. We defined already the log-likelihood. But for `optim` we need the negative log-likelihood. Also please substitute the probability with the real formula. Can we simplify the formula a bit so that we can differentiate it later?
2. To get the gradient we need to differentiate the negative log-likelihood. Please do the differentiation with respect to μ. To use it later as gradient we would need to do it also with respect to θ. After differentiation find the optimum for μ. What is it?

3.5 Running a full expression outlier detection pipeline

Usually the raw data we working with has confounders and clusters. To check if we see batches and if the data is effected by unkown confounders, visualize a row-centered correlation headmap. Hint: *rowMeans()* and *heatmap.2(x, trace='none')*.

To remove those batch effects and unkown confounders we can use existing workflows.

We can use the R-package OUTRIDER to do this and to run a full pipeline for detection of expression outliers.

```
library(OUTRIDER)
```

To show a real world example of how to solve a patient through the detection of expression ouliers run the full pipeline of outrider and visualize it. Hint *?OUTRIDER*, *?plotVolcano*, *?plotAberrantPerSample*, and *?plotExpressionRank*.