STAT 530 Bioinformatics: RNA-Seq lab

Dave Zhao

Preprocessing RNA-Seq data

In this lab we will be using a sample fastq file, `lab.fastq`, available on the course website. For the homework you will need to get fastq files from SRA, you will need to install SRA Tookit, using

```
sudo apt-get install sra-toolkit
```

Then you will run

```
fastq-dump [SRA file].sra --split-files --gzip --outdir ./fastq > [SRA file]_fastq-dump.log
```

This will produce `.fastq.gz` files. These are zipped files because fastq files are usually huge. Fortunately many of the following programs will work directly on .gz files without requiring you to unzip them.

1. Demultiplex

There are no adapters in `lab.fastq` so the first step is to demultiplex. You will need to download the fastx toolkit: `http://hannonlab.cshl.edu/fastx_toolkit/download.html`. Get the precompiled binaries and extract to your `~/bin` folder.

The reads in `lab.fastq` are multiplexed so we first need to split the reads by barcode:

```
cat lab.fastq | fastx_barcode_splitter.pl --bcfile barcodes.txt \
--bol --prefix demultiplex_ --suffix ".fastq" --mismatches 1 > demultiplex.log
```

View the log file:

```
$ cat demultiplex.log
Barcode Count Location
1 1 demultiplex_1.fastq
2 1 demultiplex_2.fastq
unmatched 0 demultiplex_unmatched.fastq
total 2
```

To remove the barcodes (the first 10 nucleotides) from each resulting file:

```
fastx_trimmer -f 11 -i demultiplex_1.fastq -o nobc_1.fastq
```

```
$ cat demultiplex_1.fastq
@HWI-EAS235_0027_FC:1:1:9931:982#0/1
GCGAGATAATGCTGGCGGCTGACGGCACCGTCATGAACACCTTCCACTCTATCCAGGGCAGGCCAGGGACTCCCTGGCCTGACACATGATGCCCA
```

```
+HWI-EAS235_0027_FC:1:1:9931:982#0/1
OOLHNNNNN_bbb_b__b_____________b_b__b___bbb______BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

$ cat nobc_1.fastq
@HWI-EAS235_0027_FC:1:1:9931:982#0/1
GCTGGCGGCTGACGGCACCGTCATGAACACCTTCCACTCTATCCAGGGCAGGCCAGGGACTCCCTGGCCTGACACATGATGCCCAGATTTCAGAT
+HWI-EAS235_0027_FC:1:1:9931:982#0/1
bbb_b__b_____________b_b__b___bbb______BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

2. Trimming

We will trim by quality using `cutadapt`, `http://cutadapt.readthedocs.io/en/stable/installation.html`. You may first need to install Python and `pip`, a package manager for python programs.

To trim by quality:

```
$ cutadapt -q 20 --quality-base=64 --minimum-length 20 -o qc_1.fastq nobc_1.fastq
This is cutadapt 1.9.1 with Python 2.7.10
Command line parameters: -q 20 --quality-base=64 --minimum-length 20 -o qc_1.fastq nobc_1.fasto
Trimming 0 adapters with at most 10.0% errors in single-end mode ...
Finished in 0.01 s (10000 us/read; 0.01 M reads/minute).

=== Summary ===

Total reads processed:                      1
Reads with adapters:                        0 (0.0%)
Reads that were too short:                  0 (0.0%)
Reads written (passing filters):            1 (100.0%)

Total basepairs processed:           104 bp
Quality-trimmed:                      65 bp (62.5%)
Total written (filtered):             39 bp (37.5%)
```

We need the `--quality-base=64` flag because these quality scores are encoded as ascii(phred quality + 64), whereas the `cutadapt` default is ascii(phred quality + 33). Some options, like the quality score of 20 and the minimum length of 20, may not be optimal.

3. Quality control

We will need `fastqc`. **Do not install using** `sudo apt-get install`. Instead download from `http://packages.ubuntu.com/yakkety/all/fastqc/download`

To do QC on our quality-trimmed fastq files:

```
fastqc qc_1.fastq
fastqc qc_2.fastq
```

The results are reported in a website; see Figure 1 for a screenshot.

We can use BLAST to figure out what the "overrepresented sequences" might be. Figure 2 contains the results for the sequence from `qc_2.fastq`.

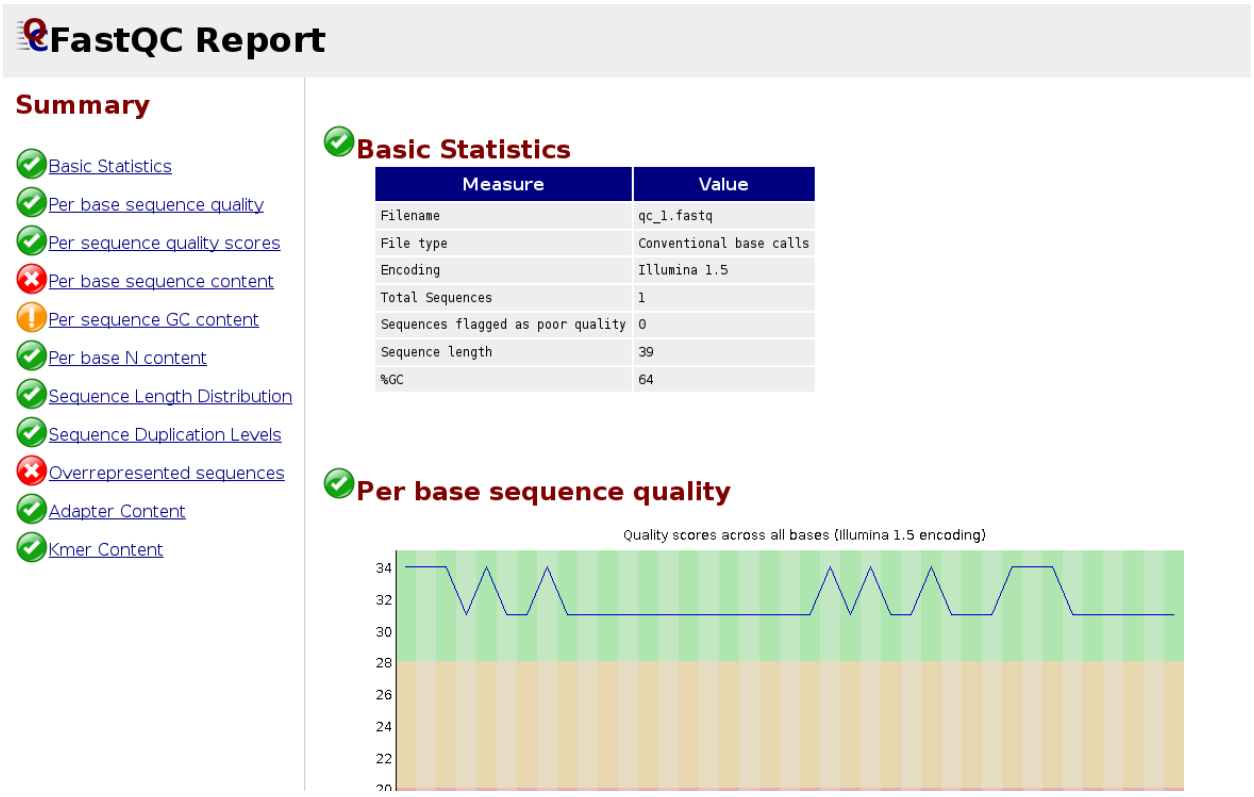

Figure 1: FastQC results

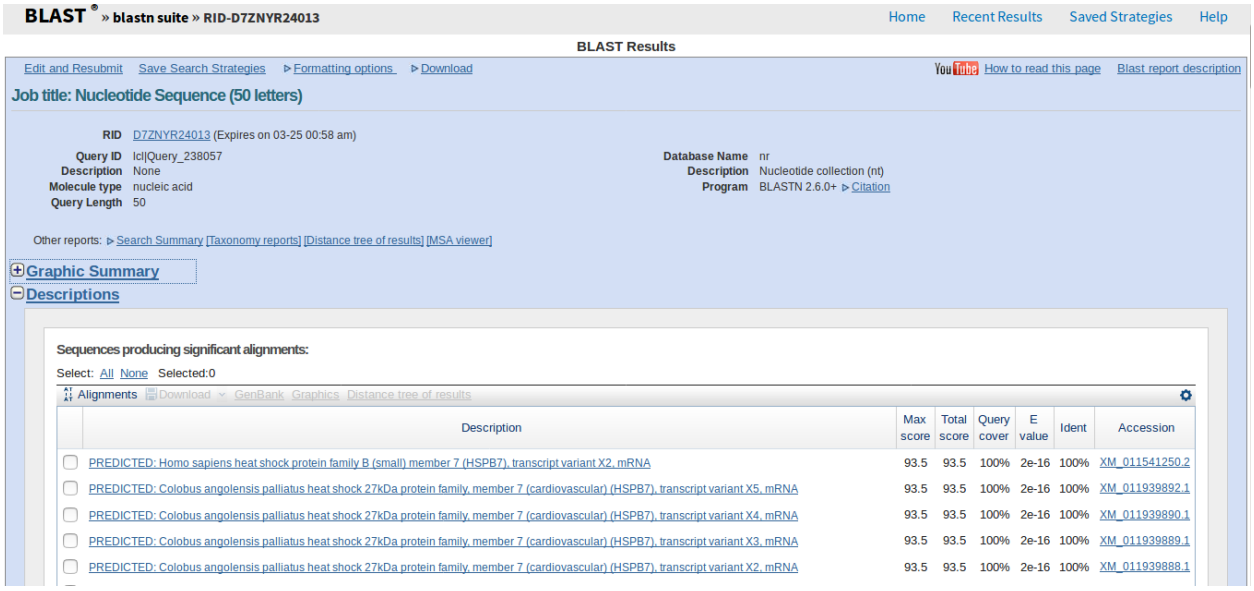

Figure 2: BLAST results

4. Aligning

For this demonstration, download `lab_fly.fastq` from the course website. This file contains RNA-seq reads from fruit fly.

We will be aligning these reads using STAR aligner. Download the pre-compiled binary `STAR` from `https://github.com/alexdobin/STAR/tree/master/bin`; use the static files if you're using Linux. After downloading, make the files executable and then make a symbolic link `STAR` and `STARlong` inside your `~/bin` directory using

```
chmod + x /home/[user name]/[STAR directory]/STAR
ln -s /home/[user name]/[STAR directory]/STAR .
```

Next we need to generate the genome index. This puts the sequence and annotation (e.g., splice junction) information into a specific data structure that allows STAR to quickly align reads to the genome. For the organism whose index we're trying to build, we need to download:

- The DNA sequence (FASTA file)
- The gene set information (GTF file)

We can get these from Ensembl: `http://useast.ensembl.org/info/data/ftp/index.html`.

The following command builds the fruit fly genome index. If you want to do this yourself you will need at least 8 GB of RAM, and will ideally have a quad-core processor.

```
STAR --runThreadN 4 --runMode genomeGenerate --genomeDir \
/home/user/STAR/Drosophila_melanogaster/ --genomeFastaFiles \
/home/user/data/genomes/Drosophila_melanogaster.BDGP6.dna.toplevel.fa \
--sjdbGTFfile /home/user/data/genomes/Drosophila_melanogaster.BDGP6.87.gtf \
--sjdbOverhang 100
```

You don't need to do this for the lab, though you won't be able to run the next few commands. On the homework, a pre-built index will be provided to you.

Next we will align the reads in `lab_fly.fastq` using the genome index we built. You will need about 4 GB of RAM to do this.

```
STAR --runThreadN 4 --genomeDir /home/user/STAR/Drosophila_melanogaster/ \
--readFilesIn lab_fly.fastq --outFileNamePrefix lab_fly \
--outSAMtype BAM SortedByCoordinate
```

This produces several files with the prefix `lab_fly`. The main one is `lab_flyAligned.sortedByCoord.out.bam` This is a **sorted** BAM file, which is necessary for most down-stream analyses.

For aligning paired-end reads: `https://groups.google.com/forum/#!topic/rna-star/mqMzbWpKRA0`

5. More quality control

After aligning it can be useful to look at statistics like percentage of mapped reads. This is in `lab_flyLog.final.out`.

6. Expression quantification

We can take the results of the alignment and count the number of reads corresponding to each fly gene. We will need to install `htseq-count` from `http://www-huber.embl.de/HTSeq/doc/install.html`. The command is

```
htseq-count -f bam -s no -i gene_id lab_flyAligned.sortedByCoord.out.bam \
/home/user/data/genomes/Drosophila_melanogaster.BDGP6.87.gtf > lab_fly.counts
```

The `lab_fly.counts` will look like

```
FBgn0000003 0
FBgn0000008 0
FBgn0000014 0
FBgn0000015 0
FBgn0000017 0
...
...
FBgn0267791 1
FBgn0267792 0
FBgn0267793 0
FBgn0267794 0
FBgn0267795 0
__no_feature 5
__ambiguous 58
__too_low_aQual 0
__not_aligned 0
__alignment_not_unique 33
```

Each row contains the counts for a gene, except the last 5 lines. Remember to remove these last five lines when doing RNA-Seq analysis.

Analyzing RNA-Seq data in edgeR

We will need to install the edgeR package in R. See `https://bioconductor.org/packages/release/bioc/html/edgeR.html`.

1. Read in data

Create a file called `targets_lab.txt` that tells you where the `.counts` files are located, as well as the covariates associated with each file:

```
file sex replicate
female_polygamous_01_gene_id.counts female 1
female_polygamous_02_gene_id.counts female 2
female_polygamous_03_gene_id.counts female 3
male_polygamous_01_gene_id.counts male 1
male_polygamous_02_gene_id.counts male 2
male_polygamous_03_gene_id.counts male 3
```

First read the targets file into R. Then read in the counts:

```
library(edgeR);
targets <- readTargets(file="targets_lab.txt");
raw_counts <- readDGE(targets$file,comment.char="_",header=F);
```

We can filter out lowly-expressed genes: require more than 5 cpm in every sample.

```
keep <- rowSums(cpm(raw_counts)>5)>=6;
counts <- raw_counts[keep,keep.lib.sizes=F];
```

After removing these genes we need to recalculate the library size. Let's take a look at what the `counts` object actually contains:

```
> names(counts)
[1] "samples" "counts"
> dim(counts$counts);
[1] 7860    6
```

Finally we can calculate TMM normalization factors: `counts <- calcNormFactors(counts);`

2. Fit negative binomial regression

Let's fit a one-way ANOVA model:

$$\log E(y_{gi} \mid sex_i) = \beta_0 + \beta_1 I(sex_i = male)$$

First we construct the design matrix for this model:

```
design <- model.matrix(~sex,data=targets);
> print(design)
  (Intercept) sexmale
1           1       0
2           1       0
3           1       0
4           1       1
5           1       1
6           1       1
attr(,"assign")
[1] 0 1
attr(,"contrasts")
attr(,"contrasts")$sex
[1] "contr.treatment"
```

We will use tagwise dispersion:

```
counts <- estimateGLMCommonDisp(counts,design);
counts <- estimateGLMTrendedDisp(counts,design);
counts <- estimateGLMTagwiseDisp(counts,design);
```

Finally we fit the GLM:

```
fit <- glmFit(counts,design,dispersion=counts$tagwise.dispersion);
> names(fit);
 [1] "coefficients"          "fitted.values"         "deviance"
 [4] "method"                "counts"                "unshrunk.coefficients"
 [7] "df.residual"           "design"                "offset"
[10] "dispersion"            "prior.count"           "samples"
[13] "prior.df"              "AveLogCPM"
> head(fit$coefficients);
            (Intercept)     sexmale
FBgn0000008  -10.053645 -0.06893862
FBgn0000017   -8.556851  0.11585496
FBgn0000018  -11.782080 -0.08882712
FBgn0000024   -9.465938  0.49491555
FBgn0000028  -11.029385 -0.09977967
FBgn0000032  -10.844203  0.01157338
```

3. Simple hypothesis tests

Let's try to identify genes that are differentially expressed between males and females in monogamous flies. This can be accomplished with a simple hypothesis test: $H_0 : \beta_1 = 0$.

```
de <- glmLRT(fit,coef=2);
```

We can find how many genes have FDR-adjusted p-value less than 0.05, as well as print the 10 genes with the smallest FDR-adjusted p-value

```
> topgenes <- decideTestsDGE(de,p.value=0.05);
> table(topgenes);
topgenes
  -1    0    1
 625 7040  195
> topTags(de,n=10,p.value=0.05);
Coefficient:  sexmale
                logFC    logCPM        LR        PValue           FDR
FBgn0019661  7.768057  9.934408 1616.3757  0.000000e+00  0.000000e+00
FBgn0005391 -7.882940 11.594502  934.3104 3.414259e-205 1.341804e-201
FBgn0004045 -8.152498 12.759892  636.5001 1.928099e-140 5.051621e-137
FBgn0004047 -7.176835 12.401382  595.7456 1.409815e-131 2.770286e-128
FBgn0034471 -2.885956  7.056330  483.0818 4.562044e-107 7.171533e-104
FBgn0038398  2.597125  7.782583  413.1770  7.457372e-92  9.769158e-89
FBgn0031925  3.001838  7.443397  326.6940  5.044901e-73  5.664703e-70
FBgn0038236  2.100567  8.495965  234.9630  4.932259e-53  4.845944e-50
FBgn0003659  1.974971  6.160634  229.9728  6.043335e-52  5.277846e-49
FBgn0040637 -3.296567  5.842501  223.1850  1.826700e-50  1.435786e-47
```

So there are 625 genes that are down-regulated in males, 195 that are upregulated in males, and the top 10 genes are displayed above.

4. Contrasts

Suppose we want to test whether $\beta_0 = \beta_1$ are the same for each gene. You'd probably never run this analysis in practice; the purpose of this example is just to demonstrate how to use contrasts in edgeR. The null hypothesis can also be expressed as $H_0 : \beta_0 - \beta_1 = 0$, so the contrast can be constructed like

```
contrast <- makeContrasts(X.Intercept.-sexmale,
                          levels=make.names(colnames(design)));
```

This can then be tested with

```
test <- glmLRT(fit,contrast=contrast);
> table(decideTestsDGE(test,p.value=0.05));
  -1    0    1
7857    2    1
```

Not surprisingly, for most of the genes $\beta_0 \neq \beta_1$.

Simulations in R

Let's do a simple simulation where we generate $X_i \sim N(\mu, 1), i = 1, \ldots, n$ and test how the accuracy of $\bar{X}$ for estimating μ behaves as a function of n. To generate the data we will set the true $\mu = 0$, consider n equal to either 10, 50, 100, 500, or 1000, and will run 200 simulations for each n. We will use squared error to measure the accuracy of $\bar{X}$.

```
n <- c(10,50,100,500,1000);
sims <- 200;
results <- matrix(NA,nrow=sims,ncol=length(n));
set.seed(1);
for(i in 1:length(n)){
  for(j in 1:sims){
    X <- rnorm(n[i],mean=0,sd=1);
    err <- (mean(X)-0)^2;
    results[j,i] <- err;
  }
}
```

We can then plot the average error over the 200 simulations as a function of n:

```
pdf(file="sim.pdf",width=6,height=6);
plot(n,colMeans(results),type="l");
dev.off();
```

This is given in Figure 3.

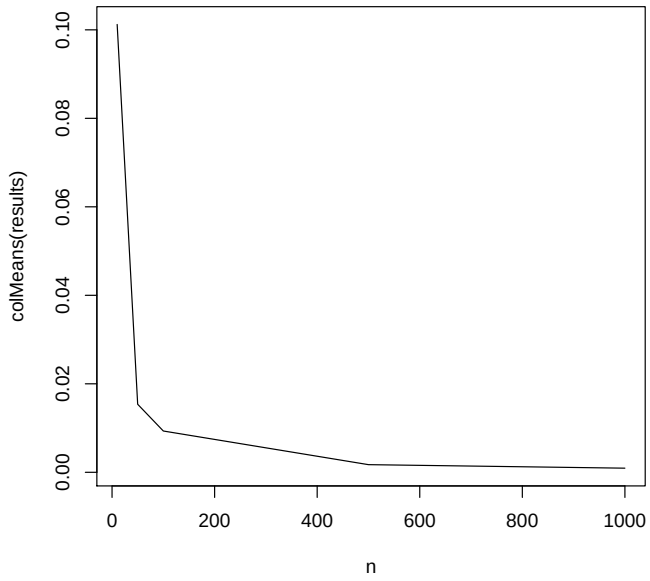

Figure 3: Simulation results