Analyzing C. elegans RNA-Seq data

Colin S. Maxwell

June 9, 2016

Contents

1 Introduction

1.1 About this document

The workflow in this paper is the one used in previous papers c.f. [Kaplan et al., 2015, Maxwell et al., 2012]. Most of the code is based on the analysis described in a previous analysis that was sent to Ryan and Jim called `Starvation_Epigenetics_and_DR_writeup.pdf`.

The overview of the workflow is:

1. Download FASTQ files from the sequencing facility
2. Map the reads using Bowtie
3. Count the number of reads that align to a gene using HTSeq
4. Test for differential expression using the package DESeq in R
5. Compute FPKM confidence intervals using the package Cufflinks
6. Test for GO enrichments, create venn diagrams, etc. using R

Unfortunately, these analysis are spread out across two different programming languages: Python and R. Python is used to create a simple pipeline to map and count reads, whereas R is used for everything else. Recently, tools such as QuasR have been developed that make the Python aspect of this document obsolete. It would be nice to re-write the pipeline to use these tools.

There will be blocks of code to run throughout the document. They will be in their own blocks, and will be in `typewriter` font. Hopefully from the context it will be clear what language to run the blocks in, but assume that they should be run in bash unless otherwise specified.

1.2 Software requirements

This document assumes that you are using the OSX operating system.

Several different programs are needed to execute all the code in this document. They should be installed according to the instructions on their website including appending the relevant code to your `.bash_profile`:

- Anaconda Python This is a distribution of Python that includes a nice package manager called `conda` as well as lots of scientific computing resourced built-in.
- Bowtie "An ultrafast memory-efficient short read aligner"
- Cufflinks "Transcriptome assembly and differential expression analysis for RNA-Seq"
- SAMtools "Samtools is a suite of programs for interacting with high-throughput sequencing data." You only need to install the program called "Samtools" and not BCFtools or HTSlib.
- IGV "The Integrative Genomics Viewer (IGV) is a high-performance visualization tool for interactive exploration of large, integrated genomic datasets"

1.3 Setting up Python

The Python script `count.py` and `map.py` are written in Python 2.7 and depend on the package `HTSeq`, which is also written in Python 2.7. This means that you may have to install an older version of Python in order to use these scripts. The easiest way to do this is with the `conda` package manager. We will create a virtual environment containing python 2.7 and then install the two packages into it (The following code blocks are taken from instructions here).

Create an environment for Python 2.7 containing `anaconda` packages:

```
conda create -n py27 python=2.7 anaconda
```

Install three packages into the new environment:

```
source activate py27 # activate the environment
conda install matplotlib
conda install -c https://conda.anaconda.org/bcbio pysam
conda install -c https://conda.anaconda.org/bcbio htseq
source deactivate # deactivate the environment
```

1.4 Setting up R

At the time of writing, my R is v3.2.3. The following R packages are required at various points in this code:

DESeq2 differential expression testing

ggplot2 plotting

biomaRt downloading genome annotations

plyr data manipulation

reshape2 data manipulation

This following code installs the non-bioconductor packages (should be run in R)

```
install.packages(c("devtools", "plyr", "reshape2", "ggplot2", "gplots"))
```

The following code installs the Bioconductor packages (should be run in R)

```
source("https://bioconductor.org/biocLite.R")
biocLite("DESeq2")
biocLite("biomaRt")
biocLite("BiocParallel")
biocLite("GO.db")
biocLite("Category")
biocLite("org.Ce.eg.db")
biocLite("GOstats")
```

1.5 Basic descriptions of data types

1.5.1 FASTA

A FASTA file is a simple way of representing either DNA or protein sequences. The FASTA 'record' begins with a `>`, and a single line with a sequence name. Subsequently, there is a long string with protein or DNA sequence following it. For example, this is the beginning of the *C. elegans* genome sequence:

```
>I
gcctaagcctaagcctaagcctaagcctaagcctaagcctaagc
ctaagcctaagcctaagcctaagcctaagcctaagcctaagcct
aagcctaagcctaagcctaagcctaagcctaagcctaagcctaa
```

The first chromosome is called `I`, and the repetitive sequence at the beginning is its telomere. **It is absolutely critical that the names of chromosomes in your genomic FASTA file match the names of the chromosomes in your annotations files!** For example, the default chromosome name of chromosome `I` when you download it from Wormbase is `CHROMOSOME_I`:

```
>CHROMOSOME_I
gcctaagcctaagcctaagcctaagcctaagcctaagcctaagc
ctaagcctaagcctaagcctaagcctaagcctaagcctaagcct
aagcctaagcctaagcctaagcctaagcctaagcctaagcctaa
```

If you try to count reads mapped to a FASTA file with different chromosome names than your annotation file, none of your reads will align to features in the chromosome!

1.5.2 FASTQ

FASTQ files are like FASTA files, except they contain information about how certain the sequencer was that the reported base was what it was. FASTQ files look like this:

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

The first line is the name of the sequence, the second line is the bases, the third line is a separator, the final line is the quality scores encoded as nonsense looking characters. In a FASTQ file generated by a sequencer, there would be millions of these records. Sadly, there are multiple versions of quality scores, so those characters can mean slightly different things. You can read about that here.

1.5.3 SAM/BAM

SAM stands for "Sequence Alignment Map" and BAM stands for "Binary Alignment Map". These are files that are produced by aligners such as Bowtie. A SAM file is a plaintext file that contains information about what a read is, where it maps to in a genome, and often the sequence of the read. A BAM file is a compressed version of a SAM file that can be 'indexed' to support 'random access'. Random access just means that you can quickly pull out all the reads that map to some particular genomic coordinates. The format is complex, but you can read about it in this article: [Li et al., 2009], or more formally here. The program `samtools` is used to convert between SAM and BAM, to index BAM files, and to do random access on BAM files.

1.5.4 GFF/GTF

A GTF file stands for "General Feature Format". It's basically as list of the names and coordinates of things like exons. There's a nice description of the format here. As an example, here's the (slightly altered) top of a GTF file that I've used before:

```
I snoRNA         exon 3747 3909 . + . exon_number "1"; gene_id "not-real"
I protein_coding exon 4119 4358 . -. exon_number "5"; gene_id "Y74C9A.3"
I protein_coding exon 4221 4358 . -. exon_number "5"; gene_id "Y74C9A.3"
I protein_coding exon 5195 5296 . -. exon_number "4"; gene_id "Y74C9A.3"
```

The data is tab delimited. It contains information about 'feature' (like exons):

- The chromosome is the record on (each of these are on chromosome `I`)
- The starting coordinate (the first record starts at coordinate 3747),
- The ending coordinate (the first record stops at 3909)
- The strand of the record (the first on is on the plus strain)
- Information about the feature such as the name of the gene it belongs to or the exon number (the first record's gene name is called 'not-real' because I changed its name)

1.5.5 'counts'

Counts are defined here as the number of shotgun reads that align within the coordinates of some feature. To get 'counts' you need 1) a genome to align reads to (a FASTA file), and 2) the locations of features in the genome. We will be using the script `htseq-count.py` to count reads that fall in the exons that make up the *C. elegans* transcriptome. You should go to the documentation for `htseq-count.py` and ponder the different ways you can define 'align within the coordinates'.

Counts are needed to do statistical testing of gene expression. They are input into the program `DESeq` that uses a statistical model based on counts to test whether the expression of a gene is different between two conditions.

1.5.6 FPKMs

FPKM stands for "Fragments Per Kilobase Million". Some people call it "RPKM" which stands for "Reads Per Kilobase Million". FPKMs are a number computed by dividing the number of counts that map to a gene by the length of the gene and the size of the library. The idea is to generate a number that represents 'how highly expressed' a gene is. A larger library size increases the number of reads mapping to a gene as does increasing the length of the gene. By normalizing out those factors, the expression level of the gene can be compared to other genes or across conditions.

FPKMs are appropriate for doing things like making plots of how a gene's expression changes or clustering data, but not for doing statistical testing. That's because by normalizing by the gene body length, you change the form of the distribution of values, which invalidates the assumptions of the statistical models such as `DESeq`.

2 Mapping

This section will run you through how to map some RNA-seq data to a *C. elegans* genome:

1. How to download the WS210 genome annotation
2. How to index the genome for use with Bowtie
3. How to map raw reads using Bowtie
4. How to check the mapping efficiency of the libraries
5. How to examine the raw data using a genome browser

2.1 Downloading reference genome and annotation

To run the code in this section, enter the bash shell and change your directory to the git repo that contains this document.

```
cd /path/to/repo
```

All of the RNA-seq experiments in the lab have been done using the coordinate system 210 from wormbase (aka WS210). That is, they are mapped to WS210.

This will download the appropriate genome FASTA file from wormbase. Note that curl pipes to sed in order to replace the chromosome names with something compatible with the annotation (and IGV).

```
curl \
 ftp://ftp.wormbase.org/pub/wormbase/species/c_elegans/sequence/genomic/c_elegans.WS210.genomic.fa.gz | \
 gunzip | \
 sed s/CHROMOSOME_//g > \
 c_elegans.WS210.genomic.fa
```

For historical reasons, all of the reads in our projects up to this point have been counted in annotations (the definitions of the bodies of genes) that belong to the WS220 genome but that have been mapped back to the WS210 genome coordinates. This is because WS220 defined lots of ncRNAs and extra transcripts that we wanted to consider in our projects. This is a little awkward, but it's worked so far. There are two GFF files that list these annotations: `WS220.modRemoved.gtf` and `GFF_for_cuffdiff.gtf`. The first file was the original one given to us by our collaborator Igor Antoshechkin that contains the WS220 annotations mapped to the WS210 coordinate system. This file, however, could not be used by the Cufflinks package because it didn't contain the appropriate attributes, so I modified it to contain them. I also cleaned up some typos where an exon that should belong to a gene was actually assigned to the neighboring gene. The result is `GFF_for_cuffdiff.gtf`.

Since we submitted the `GFF_for_cuffdiff.gtf` file for [Maxwell et al., 2012], it is in the GEO folder for that paper. This next block will download and unzip it:

```
curl \
 ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE33nnn/GSE33023/suppl/GSE33023_GFF_for_cuffdiff.gtf.gz | \
 gunzip > \
 GFF_for_cuffdiff.gtf
```

2.2 Building an indexed genome

Bowtie needs an 'indexed' version of the FASTA before it can map to it. We don't need to worry too much about what that means, but it's easy to build with our newly downloaded FASTA file. This code makes a new directory for our FASTA index called `WS210` and then builds an index called `WS210` inside of it.

```
mkdir WS210
bowtie-build c_elegans.WS210.genomic.fa WS210/WS210
```

2.3 Acquiring data

The sequencing facility provides instructions for how to download data from their servers. When you are asked to analyze RNA-seq data, follow their instructions to transfer the data to an appropriate folder on storage:

```
storage/baugh.lab/Big_Data
```

For example, two batches of RNA-seq data were downloaded into the folder called `DR_and_Starvation_Epigenetics`. The directory structure looks like this (see the sequencing facilities description as well):

```
Big_Data
---DR_and_Starvation_Epigenetics
-----Project_Bowman_2124_150107B2
-------Sample_BB1_1
-------Sample_BB2_2
...
-------Sample_BB1_12
-----Project_Bowman_2124_150108B2
-------Sample_BB1_1
-------Sample_BB2_2
...
-------Sample_BB2_12
```

The folders `Project_Bowman_2124_150107B2` and `Project_Bowman_2124_150108B2` are runs of HiSeq data. They have several samples 'pooled' together into them that have been tagged with 12 different barcodes with each barcode corresponding to a particular sample. The reads from each barcode are split across multiple FASTQ files that have been compressed using gzip:

```
Sample_BB1_1
---BB1_1_ATCACG_L001_R1_001.fastq.gz
---BB1_1_ATCACG_L001_R1_002.fastq.gz
...
---BB1_1_ATCACG_L002_R1_013.fastq.gz
Sample_BB1_2
---BB1_2_CGATGT_L001_R1_001.fastq.gz
---BB1_2_CGATGT_L001_R1_002.fastq.gz
...
---BB1_2_CGATGT_L002_R1_013.fastq.gz
```

2.4 Running the mapping script

To count these reads, we need to:

1. concatenate them into a single uncompressed FASTQ file
2. run bowtie on the FASTQ file to generate a SAM file

3. run samtools to sort the reads by their chromosomal location to allow for random access
4. convert the SAM to BAM for ease of storage and random access
5. index the BAM file for random access

Random access is important because it allows you to visualize the reads using a genome browser such as IGV. This will help you check that the reads look reasonable for an RNA-seq experiment and will help you figure out what strand the reads align to.

I wrote a python script that implements a pipeline that works for the RNA-seq data **that is returned with the directory structure above.** Note that if the sequencing facility ever changes anything about how they return their data, the script may not work. It's not user friendly in the sense that it will not test for this and any errors that are returned will be hard to interpret if you don't speak Python.

It's faster if you copy the FASTQ files to your local hard drive for mapping. Copy the folder `Project_Bowman_2124_150107B2` into the same folder as this document keeping the directory structure the same. *Note that the reads are ~15GB and the resulting BAM files are ~45GB*, so it might be necessary to clean up your hard drive a bit before running this next code.

Let's look at the documentation for the `map.py` script by calling it with the option to print its help menu. Assuming that your bash shell is in the directory of the repo, you can type this:

```
source activate py27
python map.py --help
source deactivate
```

If you do that (optional) you will see the resulting output:

```
wl-10-190-104-236:celegans_rnaseq cs$ python map.py --help
usage: map.py [-h] [-b BOWTIE_LOCATION] [-s SAMTOOLS_LOCATION]
              infolder outfolder index_location numcores

    fastq.gzs -> BAM

    This will concatenate, map, sort, binarize, and index next
    gen sequencing reads as returned by the Duke Core Facility circa 2015.

    The infolder argument should have this structure:

    infolder
    ---barcode_folder_1
    -----BB1_1_ATCACG_L001_R1_001.fastq.gz
    -----BB1_1_ATCACG_L001_R1_002.fastq.gz
    ...
    ---barcode_folder_2
    ...
    ---barcode_folder_n


positional arguments:
  infolder              Directory where fastq are
  outfolder             Directory to put the BAMS
  index_location        Location of the bowtie indexes without the stuff
                        like '.1.ebwt' on it
  numcores              Number of cores for bowtie to use

optional arguments:
  -h, --help            show this help message and exit
  -b BOWTIE_LOCATION, --bowtie_location BOWTIE_LOCATION
                        Location of the bowtie binary. If you have bowtie
                        in your PATH, you don't need to set this.
  -s SAMTOOLS_LOCATION, --samtools_location SAMTOOLS_LOCATION
                        Location of the samtools binary. If you have
samtools in you PATH, you don't need to set this
```

The help message tells you everything you need to supply to the script: the location of the data (in this case `Project_Bowman_2124_150107B2`), the folder where you want to put the BAM files that it generates (let's put them in a folder called `BAM`), the 'base name' of the indexed genome that we created (in this case `WS210/WS210`), and the number of cores the that are available on your machine (set this number as high as you can, my machine has two). There are two other optional arguments that you can use to specify

where the bowtie and samtools binaries live if you don't have them in bash `PATH` variable.

To map the reads, we just need to activate Python 2.7, call the script with the appropriate arguments, and then deactivate the Python 2.7 (for completeness):

```
source activate py27
python map.py Project_Bowman_2124_150107B2 BAM WS210/WS210 2
source deactivate
```

It will take a long time to map all of the reads. However, in the end, you should end up with a `BAM` folder that looks like this:

```
BAM
--Sample_BB1_1.bam
--Sample_BB1_1.bam.bai
--Sample_BB1_2.bam
--Sample_BB1_2.bam.bai
...
--Sample_BB1_12.bam
--Sample_BB1_12.bami
```

The `.bam` files are the actual data and the `.bami` files are the 'index' files needed to support random access.

Note that bowtie is run using the flags `--best -k 1 -m 2 -S`, which basically means report only the best unique alignments. If you don't like these flags, you will need to edit `map.py` directly.

2.5 Evaluating library quality

2.5.1 Checking mapping efficiency

Examining the mapping efficiency of an RNA-seq library is a basic way to evaluate whether it was prepared properly. Depending on the type of library, mapping efficiency should be between 80-95%. I've wrapped code to do this in a script called `run_qa.py`. If you activate the python 2.7 installation we did earlier using (`source activate py27`), once you `cd` to the directory with the script in it, you can type `python run_qa.py --help=` to see the options that you need to supply it. If you do that (optional) you will see the resulting output:

```
usage: run_qa.py [-h] infolder outfolder

 This will run idxstats from samtools on a directory of indexed bam
    files. It also prints a summary of mapping efficiency computed
    from idxstats.

positional arguments:
  infolder    Directory where the indexed bam files are.
              Doesn't work with nested directories.
  outfolder   Directory to put the reports

optional arguments:
  -h, --help  show this help message and exit
```

The script runs the program `samtools idxstats` on each BAM file in `infolder` and generates a small text file in the `outfolder` that lists the chromosome name, the length of the chromosome, the number of mapped reads mapping to it, and the number of unmapped reads mapping to it. Using this data, the script calculates the mapping efficiency for each library.

```
source activate py27
python run_qa.py -p BAM QA
source deactivate
```

Here's the first couple lines of the resulting file:

Table 1: Mapping efficiency of first four libraries as computed by `run_qa.py`

sample	total_mapped	total_unmapped	efficiency
Sample_BB1_1.bam	20141299	4987943	0.801
Sample_BB1_10.bam	22458958	4460361	0.834
Sample_BB1_11.bam	22593025	4655150	0.829
Sample_BB1_12.bam	21705088	4476974	0.829

2.5.2 "Reality checking" in a genome browser

A second way to check that the libraries have been prepared reasonably is to examine the raw reads in a genome browser. This is consistent with `The First Rule of Data Analysis`:

> "Thou shalt always examine thy data in the most raw format possible before computing summary statistics."

In this case, the summary statistic is the number of reads falling within a gene. Before we compute that, we should check that the distribution of reads can appropriately been described using a 'count'. Do the boundaries of reads tend to map well within exon boundaries? Is the library prep protocol 'stranded' (i.e. do reads mapping to a gene align to the same strand as the gene is on)? If the library is stranded, is it in the 'reverse' direction (i.e. originating from positive strand transcripts will map to the negative strand)?

I have wasted too much time by not checking these things from the outset in the past. Therefore, launch the `IGV` application. In the application, make sure you've set the genome to 'C. elegans WS220'. (For this quick check, it doesn't matter that our reads are mapped to WS210). Then simply drag a bam file onto the browser to display it. We can do this because the bam file has been indexed.

You can type gene names into the IGV browser, so let's look at the lab's 'favorite' gene daf-16. You should see something like this:

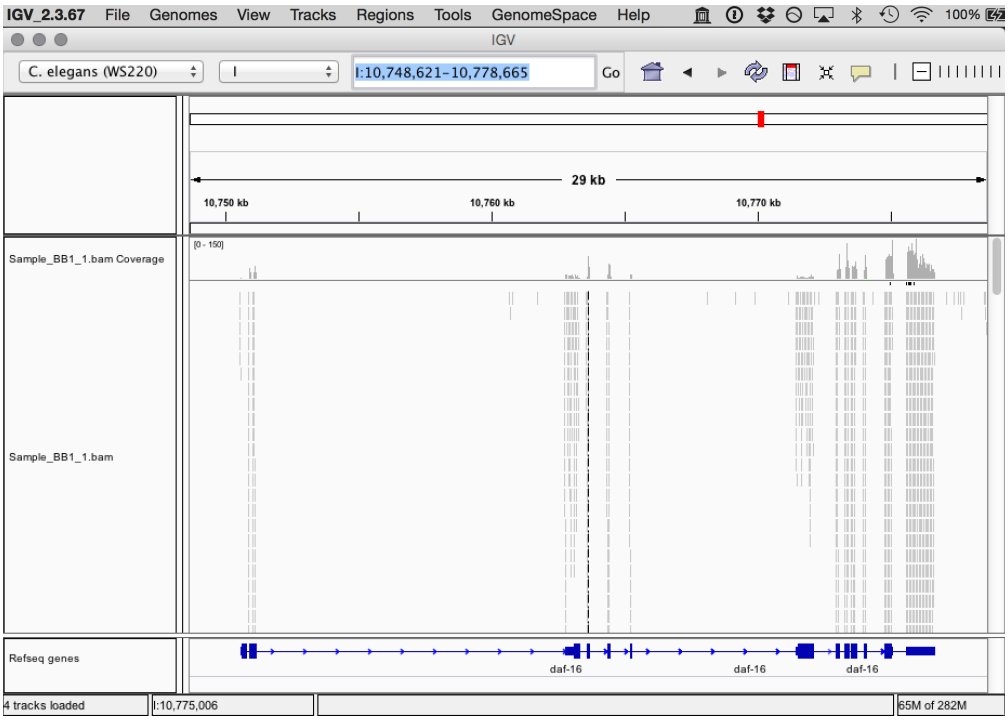

Figure 1: A screenshot in IGV of the genome window for daf-16

If you select a region around the first exon to zoom into, you can see the actual reads in the BAM file lining up in the genome:

Note that both positive and negative reads are aligning to the third exon. Conversely, only positive sense reads are aligning to the first and second exons. The fact that both positive and negative reads are aligning to the third exon suggests that the library prep was not strand specific (which is true), and the fact that the reads are aligning near

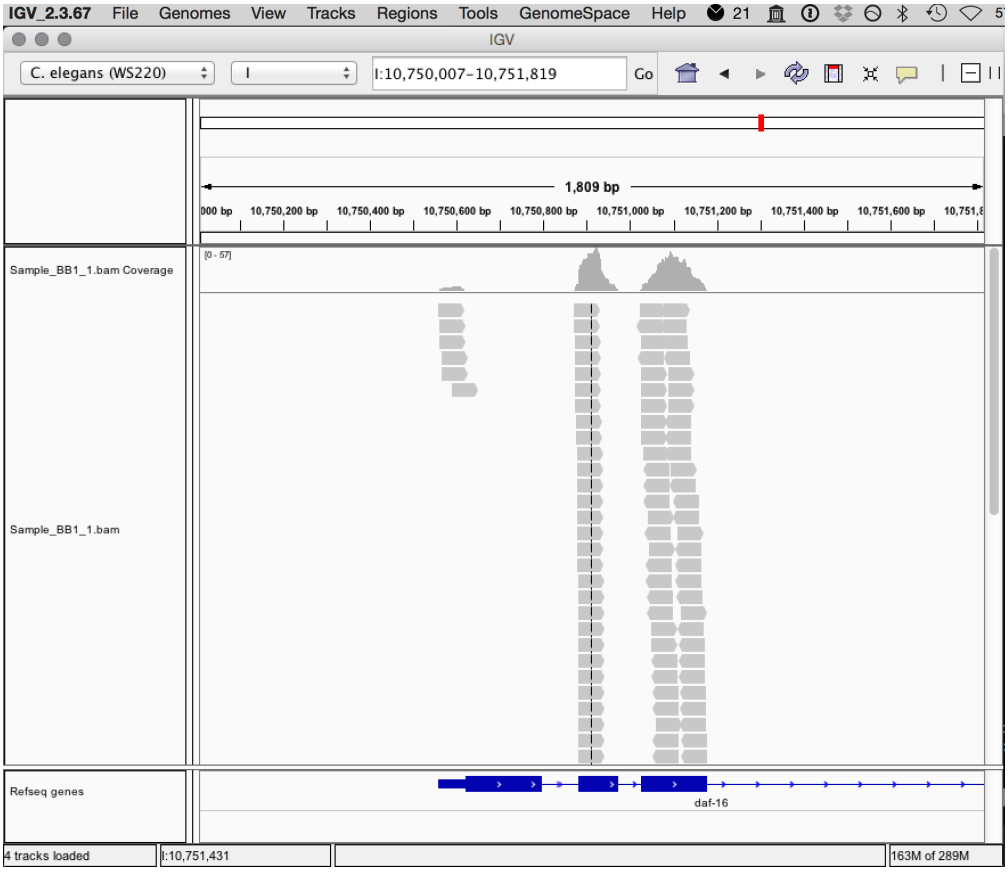

Figure 2: A screenshot in the IGV of reads lining up around the first exon of daf-16

exonic boundaries suggests that the data are in fact RNA-seq reads (which they are).

3 Counting

This section will run you through how to count reads mapping within annotated exons in the *C. elegans* genome.

3.1 Running the script

I have written a python script called `count.py` that will count the reads mapping to genes in a GTF file for each bam file in a folder. Before running the script, you should go investigate the philosophy and options for the script it is wrapped around by visiting its website.

If you are in the directory, and have activated Python 2.7, you can examine the help menu for the counting script:

```
./count.py --help
```

The arguments are similar to last time for mapping.

```
usage: count.py [-h] [-t STRANDED] infolder outfolder gtf

positional arguments:
  infolder              Directory where bam files are
  outfolder             Directory to put the count files
  gtf                   Location of the GTF file

optional arguments:
  -h, --help            show this help message and exit
  -t STRANDED, --stranded STRANDED
Is the library stranded? (This is passed directly to
the htseq-count script as its '-s' option)
```

Assuming that the bam files are in the subfolder `BAM`, that we want the counts in a folder called `counts`, and that we are using the GTF file we downloaded earlier, we can call the script like this:

```
./count.py BAM counts GFF_for_cuffdiff.gtf
```

This will take a long time to run. A way to speed this up would be to figure out how to spawn parallel shell commands in Python to count multiple libraries simultaneously, but I haven't implemented that functionality.

The output is just a bunch of text files stored in the `counts` folder. Let's look at the first 10 records of one of them:

```
head counts/Sample_BB1_1.counts
```

Which should yield this output:

```
2L52.1  287
2RSSE.1 789
2RSSE.2 162
2RSSE.3 82
2RSSE.4 0
2RSSE.5 0
2RSSE.6 26
2RSSE.8 0
3R5.1   196
3R5.2   193
```

Each gene name (e.g. `2L52.1`) has the number of reads that mapped to it. Note that these 'gene names' are technically called 'sequence names' in wormbase. You can look up what the gene is by typing it into wormbase.

We can look at the bottom of the counts:

```
tail counts/Sample_BB1_1.counts
```

Which should look like this:

```
cTel54X.1       38
cTel55X.1       544
cTel79B.1       0
cTel7X.1        0
cTel7X.3        0
__no_feature    359937
__ambiguous     525808
__too_low_aQual 0
__not_aligned   4987943
__alignment_not_unique  0
```

The 'counts' that begin with `__` refer to reads that weren't counted for one reason or another: either they didn't overlap, overlapped more than one feature, or weren't aligned to the genome (the BAM file stores both mapped and unmapped reads).

4 Calculating differential expression

This next section will analyze the count data that we generated. It will all be done in R. The R code for doing the differential expression is also available in the accompanying file deseq$_{\text{analysis.R}}$.

4.1 Downloading control data

Before you do this next section, go onto storage and go into: `Baugh_Lab_Users/Colin/bowman_2124/` and download the folder `control_counts` to the folder containing this repo. Inside it are some count files that were prepared in the same way from previous experiments.

4.2 Downloading gene ID mapping

Annoyingly, there are multiple ways to refer to the same gene, and not all mappings are 1:1. Furthermore, the annotation packages used to do GO analysis have changed since I first wrote this code. As a result, you need to download a file that maps `sequence_id` to `wormbase_id`.

```
curl \
ftp://ftp.wormbase.org/pub/wormbase/releases/WS220/species/c_elegans/annotation/geneIDs.WS220.gz | \
 gunzip > \
 c_elegans.geneIDs.csv
```

4.3 Getting set up

First, let's load the packages needed for the analysis. I'm using DESeq2 here, but previous analyses have used DESeq. The results should be similar.

```
library(DESeq2); library(RColorBrewer); library(ggplot2); library(plyr)
library(biomaRt); library(magrittr); library(reshape2);
library(BiocParallel)
BPPARAM = MulticoreParam(workers=2)
```

4.4 Reading in the metadata

To provide information about the data to R (e.g. for constructing linear models), we should construct a file that maps the file name of our data to the variables that we are testing. Bioconductor refers to these as 'phenotype' files. I have constructed one that included both the counts we created in the previous section, but also previous projects that I've worked on. It's called `pheno_big.csv`. There first three lines are shown in Table 2. To show how to test for DE, let's focus on just the data for the N2 vs. a *daf-16* mutant.

```
# read in metadata
bigPheno <- read.csv("pheno_big.csv", header=T, as.is=T)
toTest <- subset(bigPheno, stage %in% c("N2", "daf16"))
toTest$stage <- factor(toTest$stage)
```

4.5 Reading in the data

This is a file that lists whether a gene is coding or non-coding. It's extracted from WS220 using Wormmart, which no longer exists :(

```
# data file containing mapping sequence_id to coding status
load(file="geneInfo2.Rdata")
coding_genes = with(geneInfo2, sequence_name[coding_status == "coding"])
```

The DESeq package provides a convenient function to read in the data. Note that the `barcode` column is removed before reading in the data. Otherwise, `DESeq` will think your data is not replicated. We will only test the coding genes for differential expression. Note

that the 'design' argument specifies the contrast we are testing using R's usual statistical model language.

```
cds <- DESeqDataSetFromHTSeqCount(
    sampleTable = toTest,
    design = ~ stage)

cds<- cds[coding_genes,] # restrict to only coding genes
```

Finally, we'll read in the mapping between `sequence_id` and wormbase ID. Note that I removed the duplicated `sequence_ids`, since I have no way of mapping them to a gene ontology annotation.

```
id_mapping <- read.csv("c_elegans.geneIDs.csv", header=FALSE, as.is=T)
colnames(id_mapping) <- c("wormbase_id", "public_name", "sequence_id")
## Remove duplicated sequence IDs
id_mapping <- id_mapping[!duplicated(id_mapping$sequence_id),]
rownames(id_mapping) <- id_mapping[["sequence_id"]]
```

4.6 Testing for differential expression

Once the data is read in, testing for DE is simple:

```
dds <- DESeq(cds)
res <- results(dds)
write.csv(res,"DE_N2_daf16.csv")
```

The summary shows that 11% of genes were differentially expressed. Note that 14% were excluded before testing because they had very low numbers of reads mapping to them. This is done automatically by the software to maximize the true positive rate. Previously, I had manually excluded the bottom 40% of expression levels. This way of doing things is better. See the DESeq2 paper for details.

```
summary(res)
```

```
out of 19002 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up)      : 1229, 6.5%
LFC < 0 (down)    : 2002, 11%
outliers [1]      : 0, 0%
low counts [2]    : 2570, 14%
(mean count < 2)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

4.7 Testing for GO enrichments

Let's test for GO enrichments in genes that are differentially expressed with a FDR of 1%. Note that we are restricting ourselves to the universe of genes that have actually been tested for differential expression by finding genes whose test is not NA. To do this, we will use some code I wrote called `CelegansGO.R`. The test it does will test for the enrichment of 'higher' categories contingent on 'lower' categories. For details see the `GOstats` package vignette or here.

```
map_to_wb = function(seq_ids) id_mapping[seq_ids,"wormbase_id"]
## source code containing functions to do tests
source("CelegansGO.R")
## find all genes that were tested
all_genes <- rownames(res)
tested_genes <- all_genes[!is.na(res$pvalue)]
tested_genes_wb  <- map_to_wb(tested_genes)

## find all genes with sig. DE
sig_genes <- all_genes[res$padj < 0.01]
sig_genes <- sig_genes[!is.na(sig_genes)]
sig_genes_wb <- map_to_wb(sig_genes)

## calculate enrichments
enrichments <- newCelegansGO(sig_genes_wb, tested_genes_wb)
```

To examine the results, you can use the function `summary`, which will report all enrichments found with a pvalue of 0.05. However, this doesn't account for multiple testing, so instead use the function `correctedSummary`, which will report only enrichments for categories of a particular size and that meet an FDR threshold. I usually use 5 and 0.01, respectively.

Note that I've removed some columns from the output and rounded the pvalue to 3 digits for the ease of display in this automated document. Of course, in a real paper you would want to report the full p-value.

```
correctedSummary(enrichments, 5, 0.01) %>%
    transform(
Term=Term,
padjust=round(padjust,3),
OddsRatio=round(OddsRatio,1),
ExpCount = NULL,
GOBPID = NULL,
Pvalue=NULL)
```

OddsRatio	Count	Size	Term	padjust
2.4	109	551	oxidation-reduction process	0
2.6	59	272	innate immune response	0
2.5	59	280	immune system process	0
3.8	31	108	defense response to other organism	0
3.7	31	110	response to external biotic stimulus	0
3.9	28	95	response to bacterium	0
4.3	22	70	defense response to Gram-negative bacterium	0
14.8	8	13	lipid oxidation	0.001
3.9	19	64	fatty acid metabolic process	0.001
12.9	7	12	fatty acid beta-oxidation	0.007

It would be more helpful to look at enrichments that are either "up" or down. This splits the results into "up" and "down" genes, then finds the significant genes in each category. Note that according to the DESeq2 documentation:

> "For a particular gene, a log2 fold change of 1 for condition treated vs untreated means that the treatment induces a change in observed expression level of 2^1 = 0.5 compared to the untreated condition."

If you just type `res` into the command line, you'll get some output like this:

```
log2 fold change (MAP): stage daf16 vs N2
Wald test p-value: stage daf16 vs N2
DataFrame with 20270 rows and 6 columns
```

Since our data is listed as "log2 fold change (MAP): stage daf16 vs N2", a log2FoldChange greater than one means the gene is "up" in the *daf-16* mutant.

```
## The names of the list are "FALSE" and "TRUE"
res_split <- split(res, res$log2FoldChange > 0)
## Name them to "down" and "up"
names(res_split) <- c("down", "up")
## Restrict to only sig genes, remove NA, convert to WB
sig_split <- lapply(res_split,
    function(x){
out <- rownames(x)[x$padj<0.01]
out <- out[!is.na(out)]
map_to_wb(out)})
```

There are 1,308 genes that are significantly down in the *daf-16* mutant and *616* that are significantly up.

```
data.frame(num.sig=sapply(sig_split, length))
```

We can use the same functions to look for significant enrichments.

```
enrich_split <- lapply(sig_split, newCelegansGO, tested_genes_wb)
```

There is only one significant enrichment for the "down" genes, but there are many significant enrichments for the "up" genes.

```
enrich_split_summaries <- ldply(
    enrich_split,
    correctedSummary,
    5, 0.01,
    .id="direction")

transform(enrich_split_summaries,
  Term=Term,
  padjust=round(padjust,3),
  OddsRatio=round(OddsRatio,1),
  ExpCount = NULL,
  GOBPID = NULL,
  Pvalue=NULL)
```

direction	OddsRatio	Count	Size	Term	padjust
down	3.2	82	543	oxidation-reduction process	0
up	5.8	21	107	defense response to other organism	0
up	5.6	21	109	response to external biotic stimulus	0
up	3.6	34	263	innate immune response	0
up	3.4	34	271	immune system process	0
up	5.9	19	95	response to bacterium	0
up	6.3	15	70	defense response to Gram-negative bacterium	0
up	2.2	57	711	transmembrane transport	0
up	6.9	12	52	lipid catabolic process	0
up	5.6	14	72	organic anion transport	0
up	2.9	29	272	lipid metabolic process	0
up	2.5	34	356	oxoacid metabolic process	0.001
up	1.8	91	1385	single-organism localization	0.001

5 Examining correlations and mapping locations between samples

5.1 Calculating FPKMs, sample correlations, sample PCA

5.1.1 Calculating FPKMs

This next rather complicated block extracts the gene length from the GFF file used to count the reads. The annotations are read in as genomic intervals, using a file from `rtracklayer`. Overlapping exons are merged using `reduce`. The remaining code casts the data into a vector with names equal to the `sequence_id`.

```
## Calculate gene lengths
library(GenomicRanges)
library(rtracklayer)
GTF <- import.gff("GFF_for_cuffdiff.gtf", format="gtf")
grl <- split(GTF, elementMetadata(GTF)$gene_id)
grl <- reduce(grl) #merges overlapping exons
reducedGTF <- unlist(grl, use.names=T)
gene_names = rep(names(grl), elementLengths(grl))
exon_lengths = width(reducedGTF)
gene_lengths <- sapply(split(exon_lengths, gene_names),sum)
```

This will get "FPKMs" that are normalized using the DESeq library normalization protocol. The geometric mean per million thing is taken from a comment here by Simon Anders himself. Note that Simon Anders recommends using a different method to do clustering called the variance stabilizing transformation.

```
library(DESeq2)
bigPheno <- read.csv("pheno_big.csv", header=T)

## read in all the data using the DESeq convenience function
all_counts_for_fpkm <- DESeqDataSetFromHTSeqCount(
    sampleTable = bigPheno,
    design = ~ stage)

## ensure lengths match gene names
gene_lengths_coding <- gene_lengths[rownames(all_counts_for_fpkm)]

## Divide by "kilobase"
normCounts <- (counts(all_counts_for_fpkm)/
       (gene_lengths_coding/1e3))
geometricMeanPerMillion <- exp(mean(log(apply(counts(all_counts_for_fpkm),2,sum))))/1e6

## Estimate the size factors for library size scaling
all_counts_for_fpkm <- estimateSizeFactors(all_counts_for_fpkm)

## Divide by "million"
normCounts <- (normCounts/sizeFactors(all_counts_for_fpkm))/geometricMeanPerMillion
write.csv(normCounts, "all_fpkm.csv")
```

5.1.2 Looking for sample correlations

We can make a heatmap out of the data to look at which libraries have expression levels most similar to one-another (Fig. 3). Note the use of a function to generate a colorblind-friendly color scheme. Note also that replicates within experiments have good correlation and that libraries prepped from adults cluster away from the others.

```
library(gplots); library(colorRamps)
source("blue2yellow.R")

hmcol<- blue2yellow(100)

correlations <- cor(normCounts)
heatmap.2(correlations,scale="none", trace="none",
  col=hmcol, margins=c(12,12))
```

The other standard way to look at the samples you have is to do a PCA plot of the data. This next chunk will use the standard R PCA function to compute a PCA of the samples, then will merge the data with the meta data in order to make an informative(ish) plot

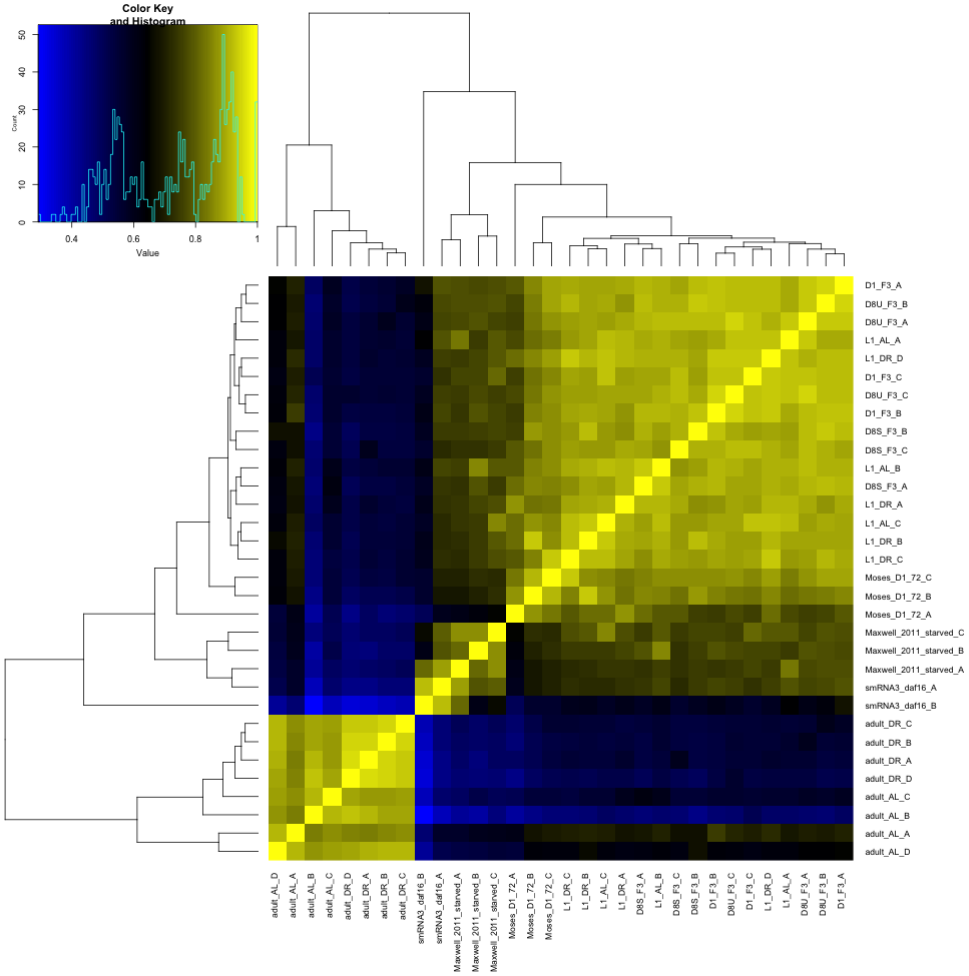

Figure 3: A heatmap of the correlation between different library types.

(Fig. 4). It looks like PC1 is separating adults from L1s.

```
library(ggplot2)
prin <- prcomp( t(normCounts))
# 'predict' the original variables into PCA space
pred  <- predict(prin, newdata=t(normCounts))

## Convert to a data.frame to allow mixed data types in
## columns
pred <- as.data.frame(pred)
pred$ID = rownames(pred)

pred <- merge(pred, bigPheno, by="ID")

# note that the colors could be any of the variables in the meta
# data file
ggplot(pred, aes(x=PC1,y=PC2,
 col=paste(stage,condition)))+
    scale_shape_manual(values=1:8)+
    geom_point()
```

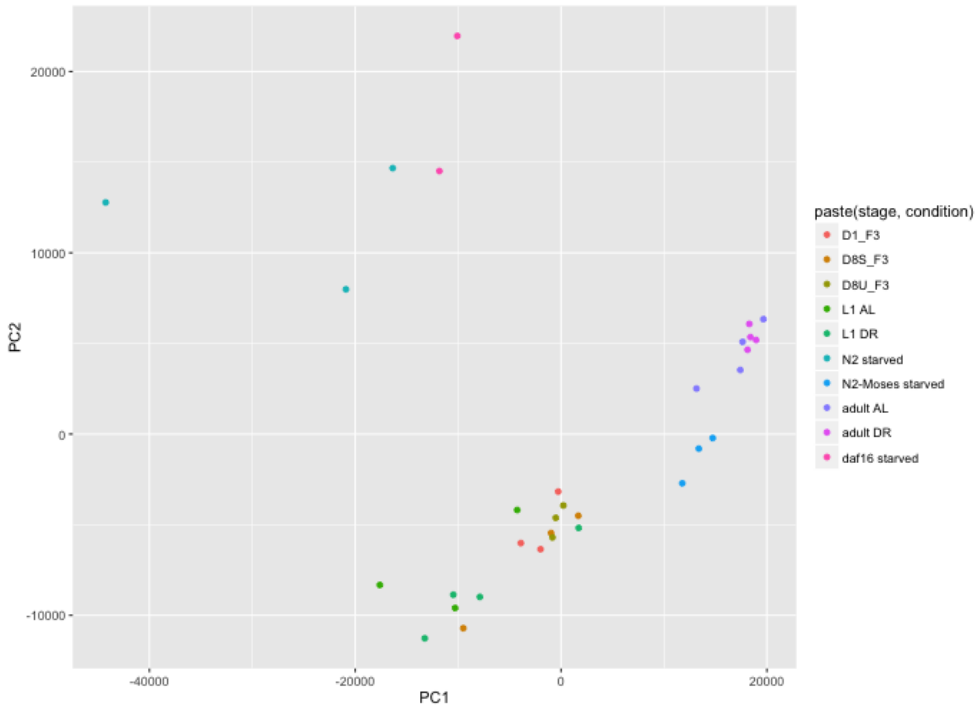

Figure 4: A PCA plot of the different conditions we have counts for colored by what stage and condition they are

5.2 Heat maps for genes

```
library(reshape2); library(magrittr)


res=read.csv("DE_N2_daf16.csv", header=T, as.is=T, row.names=1)

sig_genes <- rownames(res)[(res$padj<0.01) & (abs(res$log2FoldChange) > 2)]
sig_genes <- sig_genes[!is.na(sig_genes)]

normCounts <- read.csv("all_fpkm.csv", header=T, as.is=T, row.names=1)
normCounts <- subset(normCounts, rownames(normCounts) %in% sig_genes)
#normCounts <- log2(normCounts/(1+apply(normCounts, 1, mean)))


tmp <- counts(cds)
tmp <- subset(tmp, rownames(cds) %in% sig_genes)

library(Heatplus)
source("blue2yellow.R")


pl <- annHeatmap2(as.matrix(normCounts),
  col=blue2yellow,
  cluster=list(dendro=list(rows="yes", col="yes")),
  data.frame(
      row.names=colnames(counts),
      gen=c("WT", "WT","WT","mut", "mut")))
plot(pl)
```

```
bigPheno <- read.csv("pheno_big.csv", header=T, as.is=T, row.names=1)

normScaled <- scale(normCounts,center=TRUE, scale=FALSE)

sm_heatmap <- annHeatmap2(normScaled,
 bigPheno,
 scale="none",
 dendrogram=list(dendro=list(rows="yes",cols="yes")))
plot(sm_heatmap)
```

5.3 Looking where reads map

This is some code that will split up the libraries to look at where reads are mapping by gene type. The expectation is that poly-A libraries should have less mapping to ncRNAs.

```
load(file="geneInfo2.Rdata")
coding_genes = with(geneInfo2, sequence_name[coding_status == "coding"])

## use convenience function to read in all data
## extract counts
all_counts <- DESeqDataSetFromHTSeqCount(
    sampleTable = bigPheno,
    design = ~ stage) %>%
    counts() %>%
    melt()

## rename columns for easy merging
colnames(all_counts) <- c("sequence_name", "ID", "count")
all_counts <- merge(all_counts, bigPheno, by="ID")
all_counts <- merge(all_counts, geneInfo2, by="sequence_name")

## cast to look at library by type
counts_by_type <- ddply(all_counts,
c("project", "stage",
  "condition", "ID",
  "coding_status"),
plyr::summarize,
total = sum(count, na.rm=T))
counts_by_type <- ddply(counts_by_type,
    c("project", "stage",
      "condition", "ID"),
    plyr::mutate,
    percent = total/sum(total))
```

Make a quick plot of where the reads are mapping (Fig. 5). Consistent with expectation the libraries prepared with poly-A selection rather than ribominus have fewer non-coding reads.

```
p = ggplot(counts_by_type,
       aes(x=ID,y=percent,fill=coding_status))+
    geom_bar(stat="identity")+
    facet_wrap(~project, scale="free_x")+
    theme(axis.text.x=element_text(angle = 45, hjust = 1),
  legend.position="bottom")
print(p)
```

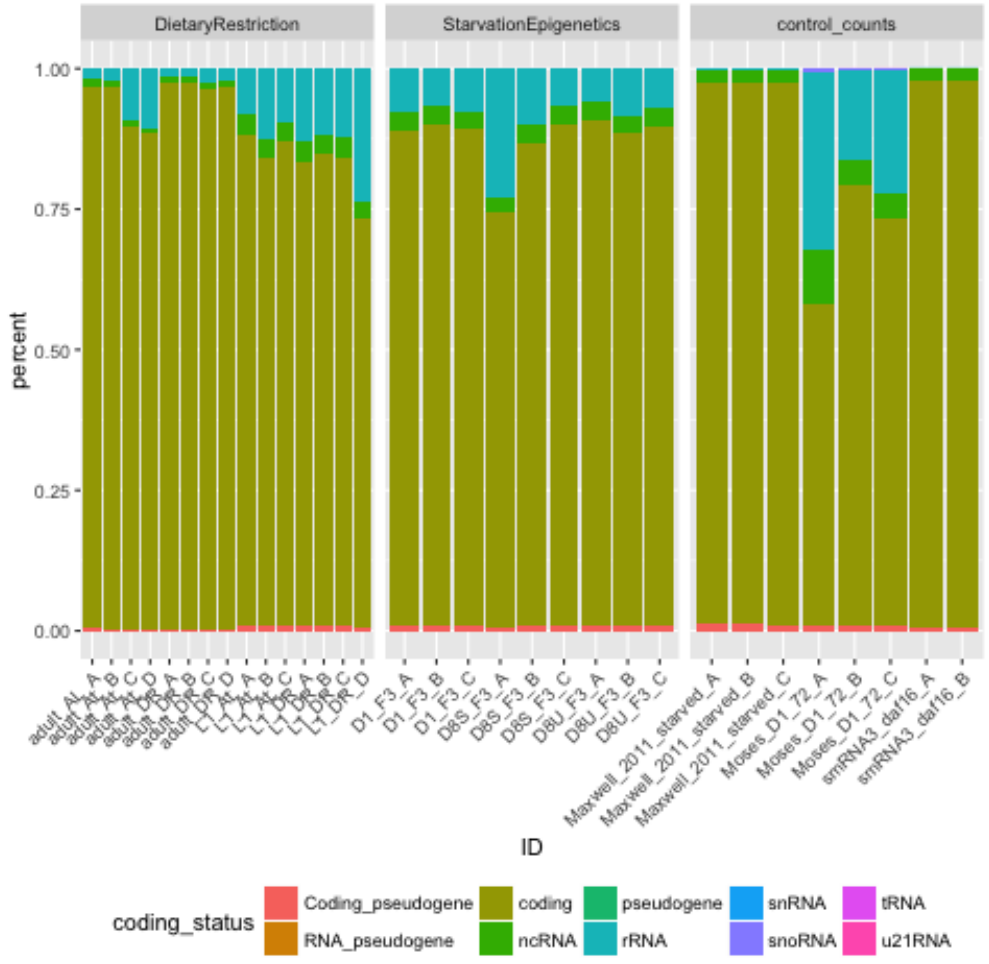

Figure 5: A barplot of where reads are mapping in the different libraries.

References

[Anders and Huber, 2010] Anders, S. and Huber, W. (2010). Differential expression analysis for sequence count data. *Genome biol*, 11(10):R106.

[Kaplan et al., 2015] Kaplan, R. E., Chen, Y., Moore, B. T., Jordan, J. M., Maxwell, C. S., Schindler, A. J., and Baugh, L. R. (2015). dbl-1/tgf-β and daf-12/nhr signaling mediate cell-nonautonomous effects of daf-16/foxo on starvation-induced developmental arrest. *PLoS Genet*, 11(12):e1005731.

[Li et al., 2009] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., et al. (2009). The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079.

[Maxwell et al., 2012] Maxwell, C., Antoshechkin, I., Kurhanewicz, N., Belsky, J., and Baugh, L. R. (2012). Nutritional control of mRNA isoform expression during developmental arrest and recovery in C. elegans. *Genome Research*, 22(10):gr.133587.111–1929.

6 Appendix

Table 2: An example of a metadata file. This is the first three rows of the table `pheno_big.csv`

ID	fn	project	stage	condition	barcode
Maxwell_2011_starved_A	control_counts/BC2.GnTn.WS210.Ensembl.bam.counts	control_counts	N2	starved	Maxwell_2011_starved_A
Maxwell_2011_starved_B	control_counts/BC5.GnTn.WS210.Ensembl.bam.counts	control_counts	N2	starved	Maxwell_2011_starved_B
Maxwell_2011_starved_C	control_counts/BC13.GnTn.WS210.Ensembl.bam.counts	control_counts	N2	starved	Maxwell_2011_starved_C