seqbias
Assessing and Adjusting for Technical Bias in High Throughput Sequencing Data

Daniel Jones
<dcjones@cs.washington.edu>
Computer Science & Engineering
University of Washington

October 17, 2016

1 Introduction

This package is designed as a means to assess and adjust for technical bias in high-throughput sequencing datasets, RNA-Seq being a specific target. As noted in previous studies, RNA-Seq is often subject to protocol specific bias. That is, the number of reads mapping to a particular position of the genome is dependent on the the surrounding nucleotide sequence (as well as the abundance of the RNA transcript) [**?**] [**?**]. Accounting for this bias increases uniformity of coverage and may result in more accurate quantification.

The approach implemented here trains a simple Bayesian network classifier and uses it to evaluate the per position bias. This builds off work done by Hansen, et. al. [**?**], available in the `Genominator` Bioconductor package. Another approach is taken by Li, et. al. [**?**] in the `mseq` package, available from CRAN.

For this vignette, we will use some example data taken from Mortazavi, et. al. [**?**] (NCBI accession number SRR001358). Because of space constraints, we have mapped the reads (using Bowtie [**?**]) to an artificial genome consisting of approximately 100kb of exonic DNA.

This "artificial genome" is given as,

```
> library(seqbias)
> library(Rsamtools)
> ref_fn <- system.file( "extra/example.fa", package = "seqbias" )
> ref_f <- FaFile( ref_fn )
> open.FaFile( ref_f )
```

And the mapped reads,

```
> reads_fn <- system.file( "extra/example.bam", package = "seqbias" )
```

2 Assessment

As a natural first step, we would like to assess whether our sample is significantly biased. If this proves to be the case, we may wish to correct for this. A simple procedure to do so will be covered in the next section.

To assess the nucleotide frequency we will use a very simple procedure:

1. Generate a random sample of intervals across our reference genome.
2. Extract sequences for these intervals from a FASTA file.
3. Extract read counts across these intervals from a BAM file.
4. Using these sequences and counts, compute and plot nucleotide or k-mer frequencies.

Sampling

For this step, we could use collection of known exons, but trustworthy annotations are not always available, and biasing the analysis by known exons may be a concern in some instances. Fortunately, `seqbias` provides a function to generate random intervals.

First, we extract a vector of sequence lengths, in the reference genome. Given an FASTA file than has been indexed with the `samtools faidx` command, we can use the `Rsamtools` package to read off the sequence lengths and to extract the sequence. First, the lengths,

```
> ref_seqs <- scanFaIndex( ref_f )
```

Once we have this, we generate 5 intervals of 100kb. It most cases we would want to generate a larger sample, but since we are working here with small reference sequence with dense coverage, we can get an accurate measurement with a few intervals.

```
> I <- random.intervals( ref_seqs, n = 5, m = 100000 )
```

Sequences

Next we extract the nucleotide sequences,

```
> seqs <- scanFa( ref_f, I )
```

The `scanFa` function does not respect strand, so we must be sure to perform the reverse complement ourselves.

```
> neg_idx <- as.logical( I@strand == '-' )
> seqs[neg_idx] <- reverseComplement( seqs[neg_idx] )
```

Counts

Finally, we count the number of reads mapping to each position in our sampled intervals.

```
> counts <- count.reads( reads_fn, I, binary = T )
```

Unless the `binary` argument is FALSE, this function returns a 0-1 vector, where a position is 0 if no reads map to it, and 1 if *at least* one read maps to it. This is a more robust way to measure sequencing bias, as the frequencies can not get dominated by a few very high peaks.

Frequencies

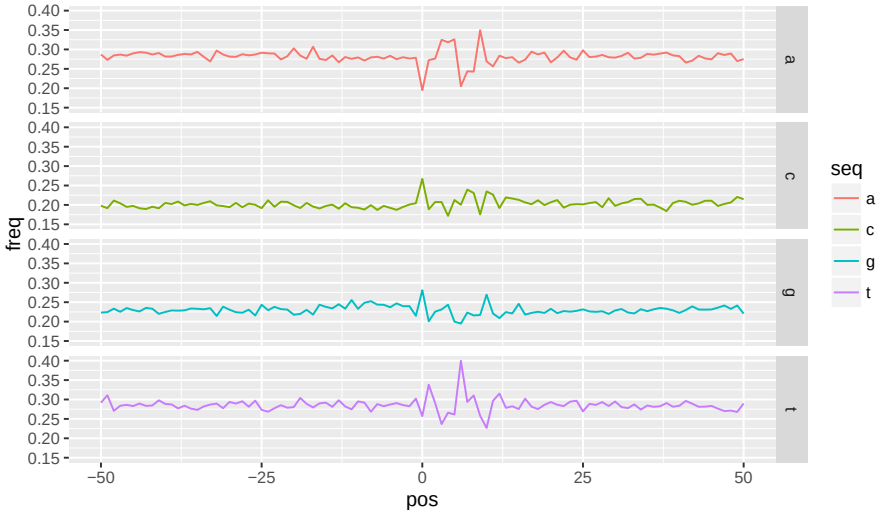
At last, we compute the k-mer frequency (where $k = 1$, by default).

```
> freqs <- kmer.freq( seqs, counts )
```

A nice way to plot this is with the `ggplot2` package, if available.

```
> if( require(ggplot2) ) {
+   P <- qplot( x = pos,
+               y = freq,
+               ylim = c(0.15,0.4),
+               color = seq,
+               data  = freqs,
+               geom  = "line" )
+   P <- P + facet_grid( seq ~ . )
+   print(P)
+ } else {
+   par( mar = c(5,1,1,1), mfrow = c(4,1) )
+   with( subset( freqs, seq == "a" ),
+         plot( freq ~ pos, ylim = c(0.15,0.4), sub = "a", type = 'l' ) )
+   with( subset( freqs, seq == "c" ),
+         plot( freq ~ pos, ylim = c(0.15,0.4), sub = "c", type = 'l' ) )
+   with( subset( freqs, seq == "g" ),
+         plot( freq ~ pos, ylim = c(0.15,0.4), sub = "g", type = 'l' ) )
+   with( subset( freqs, seq == "t" ),
+         plot( freq ~ pos, ylim = c(0.15,0.4), sub = "t", type = 'l' ) )
+ }
```

Doing so will produce the following plot,



The x-axis shows the nucleotide position relative to the read start. Negative number occur in the genome to the left of mapped reads. In this set, the reads consist of positions 0-24.

We can see a clear bias here in positions 0-15, approximately. The rest of the plot looks relatively flat, as we would expect if the experiment was measuring abundance only and not a biased by the nucleotide sequence. In the next section we will adjust read counts to account for this.

3 Compensation

Training

To begin, we must fit a seqbias model to our dataset. This done very easily with the `seqbias.fit` function. This will take only a few seconds, but when more reads are available a more accurate model can be trained at the expense of the training procedure taking up to several minutes.

```
> sb <- seqbias.fit( ref_fn, reads_fn, L = 5, R = 15 )
```

The `L` and `R` arguments control the maximum number of positions to the left and right of the read start that may be considered. The model tries to consider only informative positions, so increasing these numbers will increase training time, but should never have a negative effect the accuracy of the model.

Prediction

Once we have trained the seqbias model, we can use it to predict the sequencing bias across a set of intervals.

```
> bias <- seqbias.predict( sb, I )
```

Adjustment

To adjust, we will can simply divide the `counts` vectors by the `bias` vectors.

```
> counts.adj <- mapply( FUN = `/`, counts, bias, SIMPLIFY = F )
```

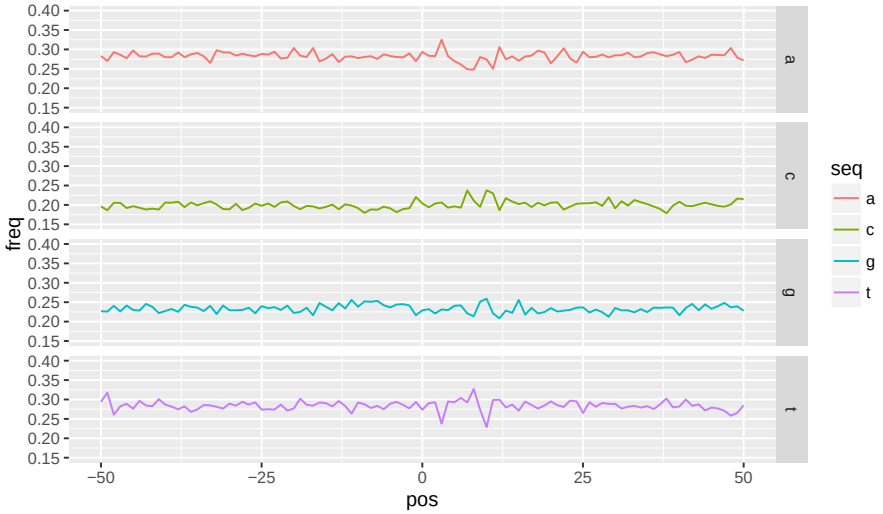
The post-adjustment nucleotide frequencies can then be measured as before,

```
> freqs.adj <- kmer.freq( seqs, counts.adj )
```

And plotted,

```
> if( require(ggplot2) ) {
+   P <- qplot( x = pos,
+               y = freq,
+               ylim = c(0.15,0.4),
+               color = seq,
+               data  = freqs.adj,
+               geom  = "line" )
+   P <- P + facet_grid( seq ~ . )
+   print(P)
+ } else {
+   par( mar = c(5,1,1,1), mfrow = c(4,1) )
+   with( subset( freqs.adj, seq == "a" ),
+         plot( freq ~ pos, ylim = c(0.15,0.4), sub = "a", type = 'l' ) )
+   with( subset( freqs.adj, seq == "c" ),
+         plot( freq ~ pos, ylim = c(0.15,0.4), sub = "c", type = 'l' ) )
+   with( subset( freqs.adj, seq == "g" ),
+         plot( freq ~ pos, ylim = c(0.15,0.4), sub = "g", type = 'l' ) )
+   with( subset( freqs.adj, seq == "t" ),
+         plot( freq ~ pos, ylim = c(0.15,0.4), sub = "t", type = 'l' ) )
+ }
```

The plot below results,



Compared to the first figure, the improvement is clear.

4 Save / Load

If the model is fit using a large number of reads, it can take several minutes to train. To avoid repeatedly refitting the model, `seqbias` provides a mechanism to save and load the model to a YAML file with the `seqbias.save` and `seqbias.load` functions.

```
> seqbias.save( sb, "my_seqbias_model.yml" )
> # load the model sometime later
> sb <- seqbias.load( ref_fn, "my_seqbias_model.yml" )
```

Note when loading the model, we need to provide a reference sequence. The `seqbias` object keeps track of the reference sequence to make `seqbias.predict` more convenient.

5 Session Info

```
> sessionInfo()

R version 3.3.1 (2016-06-21)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.9.5 (Mavericks)

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
```

```
attached base packages:
[1] parallel  stats4    stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] ggplot2_2.1.0        Rsamtools_1.26.0     seqbias_1.22.0
 [4] Biostrings_2.42.0    XVector_0.14.0       GenomicRanges_1.26.0
 [7] GenomeInfoDb_1.10.0  IRanges_2.8.0        S4Vectors_0.12.0
[10] BiocGenerics_0.20.0

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.7         magrittr_1.5        zlibbioc_1.20.0    munsell_0.4.3
 [5] BiocParallel_1.8.0 colorspace_1.2-7    stringr_1.1.0      plyr_1.8.4
 [9] tools_3.3.1         grid_3.3.1          gtable_0.2.0       digest_0.6.10
[13] reshape2_1.4.1      bitops_1.0-6        labeling_0.3       stringi_1.1.2
[17] scales_0.4.0
```