Using the qrqc package to gather information about sequence qualities

Vince Buffalo

Bioinformatics Core
UC Davis Genome Center

vsbuffalo@ucdavis.edu

2012-02-19

Abstract

Many projects in bioinformatics begin with raw sequences from a high-throughput sequencer that must be quality checked before additional analysis steps can proceed. The *qrqc* (Quick Read Quality Control) package is a fast and extensible package that reports basic quality and summary statistics on FASTQ and FASTA files, including base and quality distribution by position, sequence length distribution, k-mers by position, and common sequences.

1 Reading in a Sequence File

The *qrqc* package reads and processes FASTA and FASTQ files in C for speed, through the function `readSeqFile`. Optionally, sequences can be hashed (also done at the C level), to see the most frequent sequences in the file. Hashing is memory intensive (all unique sequence reads are kept in memory once), so by default, `readSeqFile` will randomly sample 10% of the reads and hash these. This proportion can be adjusted with the `hash.prop`. Also, k-mers are hashed (when `kmer=TRUE`) with the same proportion as set by `hash.prop`.

```
> library(qrqc)
> s.fastq <- readSeqFile(system.file('extdata', 'test.fastq', package='qrqc'))
```

Note that there is a maximum sequence length argument in `readSeqFile`, `max.length`. By default, this is 1,000. It is used to pre-allocate the matrices in C, and could be much larger than the largest sequence encountered without many downsides (its memory usage is relatively low). If a sequence larger than `max.length` is encountered, the function will stop, and the user can call the function again with a larger `max.length`.

`Readseqfile` produces a `FASTQSummary` object, which inherits from the `SequenceSummary` class. Printing the object lists a very short summary:

```
> s.fastq
```

```
Quality Information for: test.fastq
 100 sequences, 88 unique with 0.10 being sampled
```

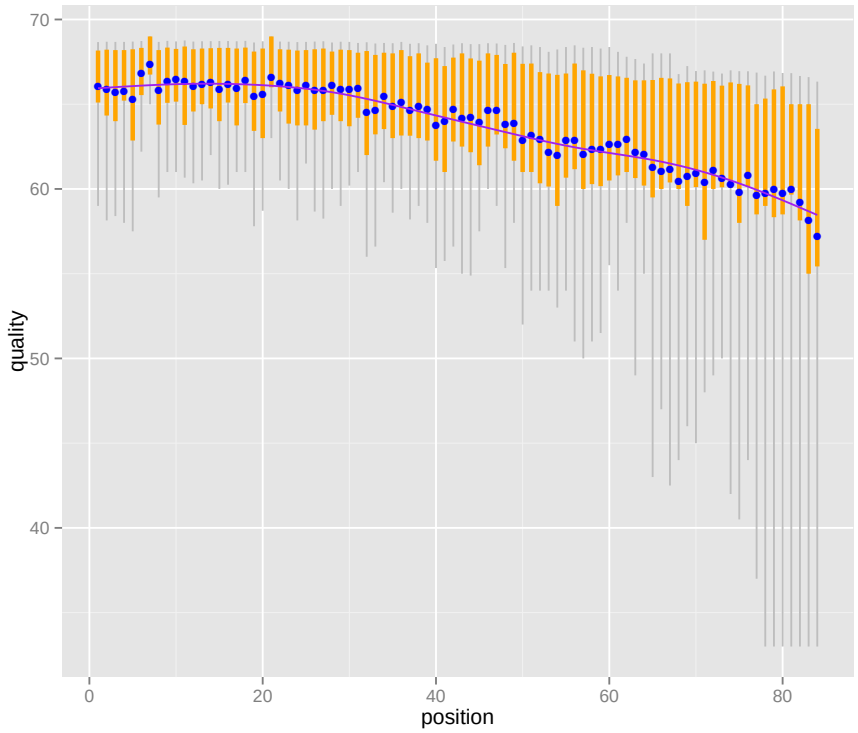

Figure 1: A plot of quality by base position, with sequence length histogram.

```
mean quality: 63.673900
min sequence length: 84
max sequence length: 84
```

Optionally, `readSeqFile` can be run without hashing, with `hash=FALSE`. `readSeqFile` also works on FASTA files, but `type=FASTA` must be specified.

2 Plotting Quality of FASTQSummary Objects

If the file read and summarized with `readSeqFile` is a FASTQ file (and thus the resulting object is from the `FASTQSummary` class), quality information by position can be plotted with `qualPlot`, which produces a graphic as in Figure **??**:

```
> qualPlot(s.fastq)
```

If there's variability in sequence length, one should interpret this quality plot with the sequence length histogram (as produced by `seqlenPlot`), as low quality at a particular base position is less worrisome if there are few reads with this sequence length. The grey lines indicate the 10% and 90% quantiles, orange lines indicate the lower and upper quartiles, the blue dot is the median, and the green dash the mean. A purple smooth curve is fit through the distributions. This line is fit by first randomly drawing values from the empirical (binned) distribution of qualities at a particular base, then using *ggplot2*'s `geom_smooth` with these points.

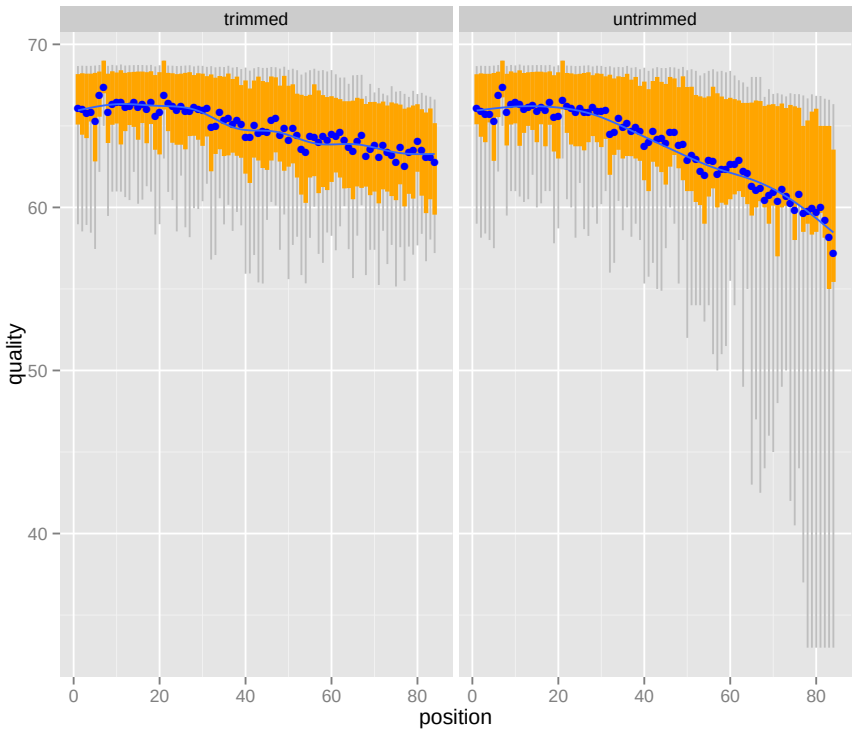

Figure 2: A plot of quality by base position after being trimmed with `Sickle`.

`qualPlot` can be very useful in inspecting base qualities before and after read quality control pipelines. For example, the package contains `test-trimmed.fastq`, which has the same sequences as `test.fastq` after being trimmed with Nik Joshi's `Sickle`, a windowed adaptive quality trimmer.

Each plot can be made separately by passing each `FASTQSummary` object directly to `qualPlot`. However, `qualPlot` has methods to handle a named list of `FASTQSummary` objects. Each will be plotted in a panel of its own.

```
> s.trimmed.fastq <- readSeqFile(system.file('extdata', 'test-trimmed.fastq', package='qrqc'))
> qualPlot(list("trimmed"=s.trimmed.fastq, "untrimmed"=s.fastq))
```

3 Base Plots of FASTQSummary and FASTASummary Objects

`qualPlot` is the only plotting method that works only on `FASTQSummary` objects. Other plotting methods work for FASTQSummary and FASTASummary objects. Base frequencies (counts) and base proportions by position can be plotted with `basePlot`. When used with `type='frequency'`, `basePlot` produces a graphic as in Figure 3:

```
> basePlot(s.fastq)
```

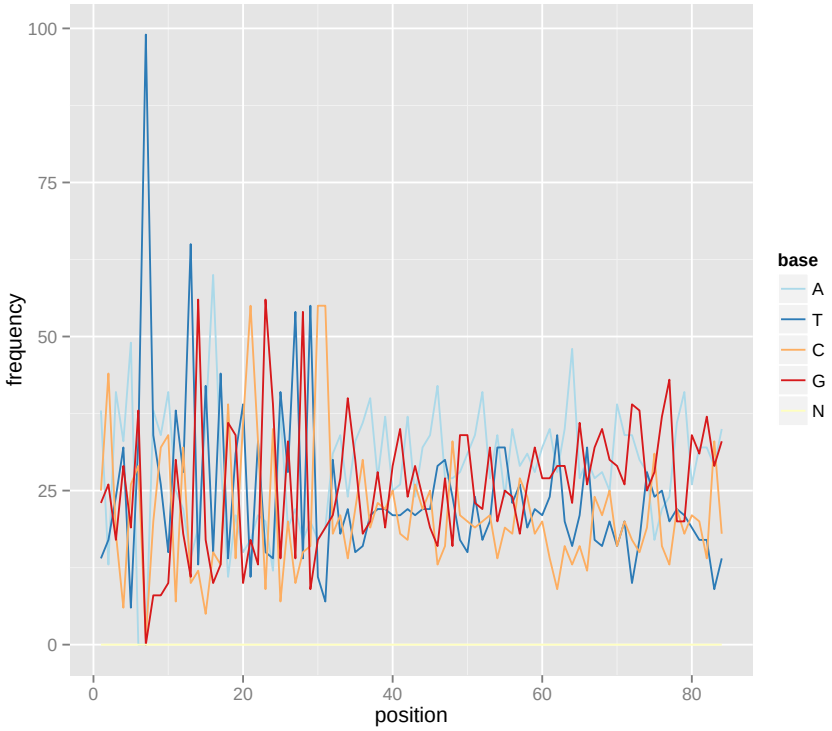

Figure 3: Base frequencies by position in sequence.

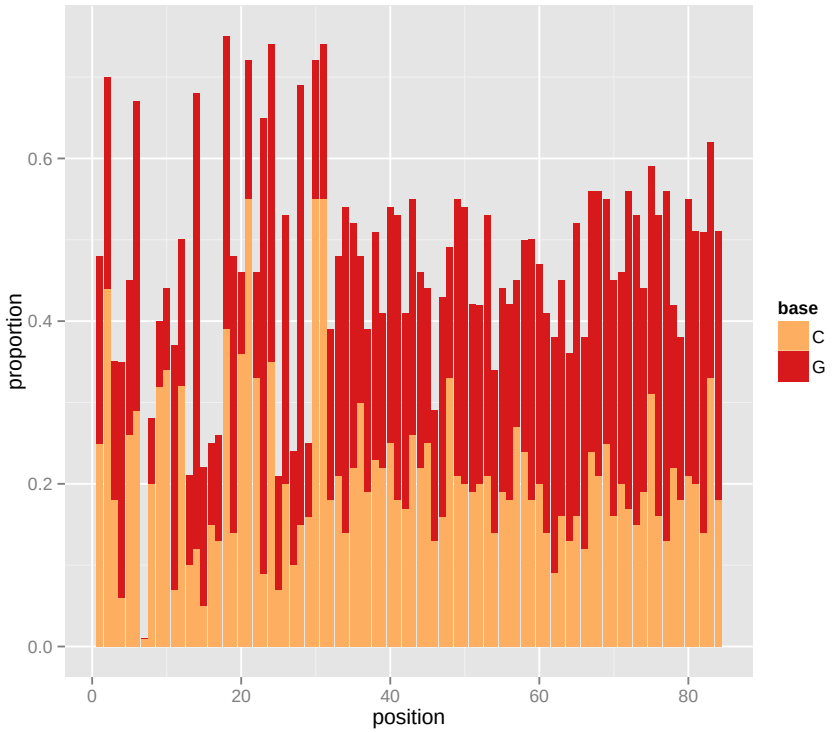

Figure 4: Base proportions by position in sequence.

`basePlot` uses a color scheme from *biovizBase*, which (according to their goals) make biological sense, are aesthetically pleasing, and accommodate those that are colorblind.

By default, the color scheme is determined by *biovizBase*'s `getBioColor` with "DNA_BASES_N" (which includes A, T, C, G, and N). A scheme for IUPAC codes can be used too; use `colorvalues=getBioColor('IUPAC_CODE_MAP')`. If one is making custom graphics, these color schemes can be more easily accessed through `scale_color_dna` and `scale_color_iupac`.

`basePlot` also accepts a `bases` parameter, which can be used to specify specific bases. This is useful for plotting just the frequency of 'N'.

Base proportions by position can be plotted with `basePlot`, with `type='proportion'`. This plot is basically identical to the plot produced with `type='frequency'` with a different y scale. Different geoms (see *ggplot2* for more details) can be specified too with the `geom` parameter. By default, a line graph is used, but bar graphs (through `geom='bar'`) and bar graphs with dodged bars (through `geom='dodge'`) can be plotted also, as in Figure 4.

```
> basePlot(s.fastq, bases=c("G", "C"), geom="bar", type="proportion")
```

Sequence length distribution can be plotted with `seqlenPlot` (graphic shown in Figure **??**):

```
> seqlenPlot(s.trimmed.fastq)
```

The GC content can be plotted with `gcPlot` (graphic shown in Figure **??**). Here, we can start to see how using *ggplot2* can easily add to *qrqc*'s plotting functions.

```
> gcPlot(s.fastq) + geom_hline(yintercept=0.5, color="purple")
```

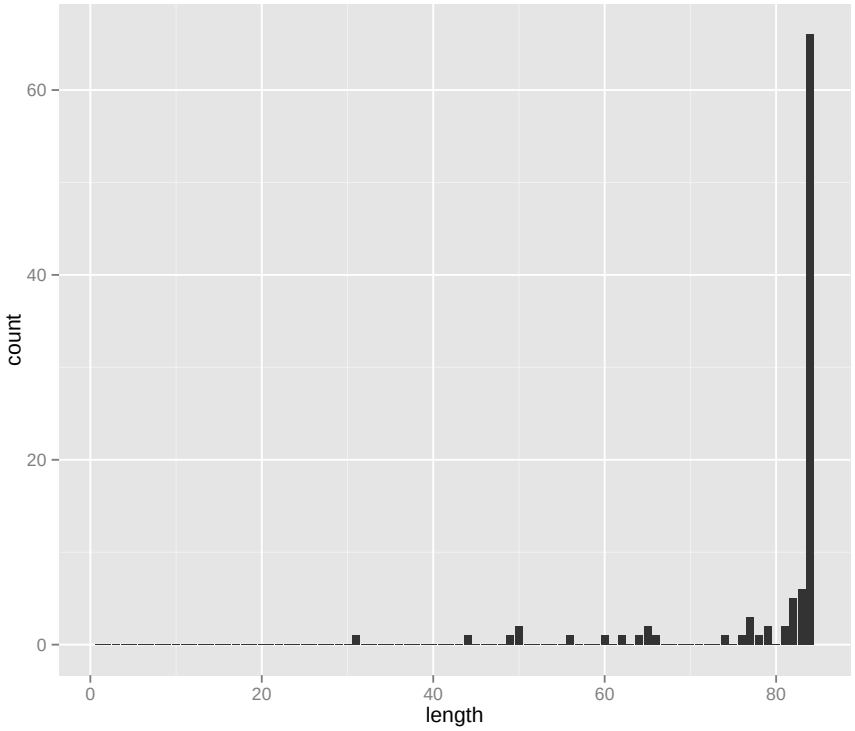

Figure 5: Histogram of sequence lengths after quality trimming.

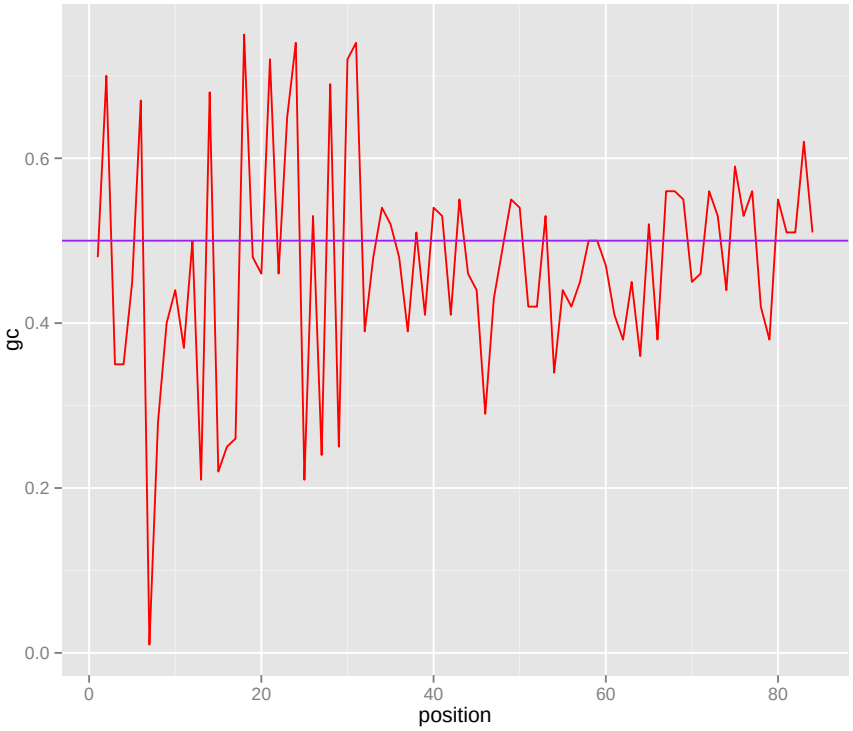

Figure 6: GC content by position

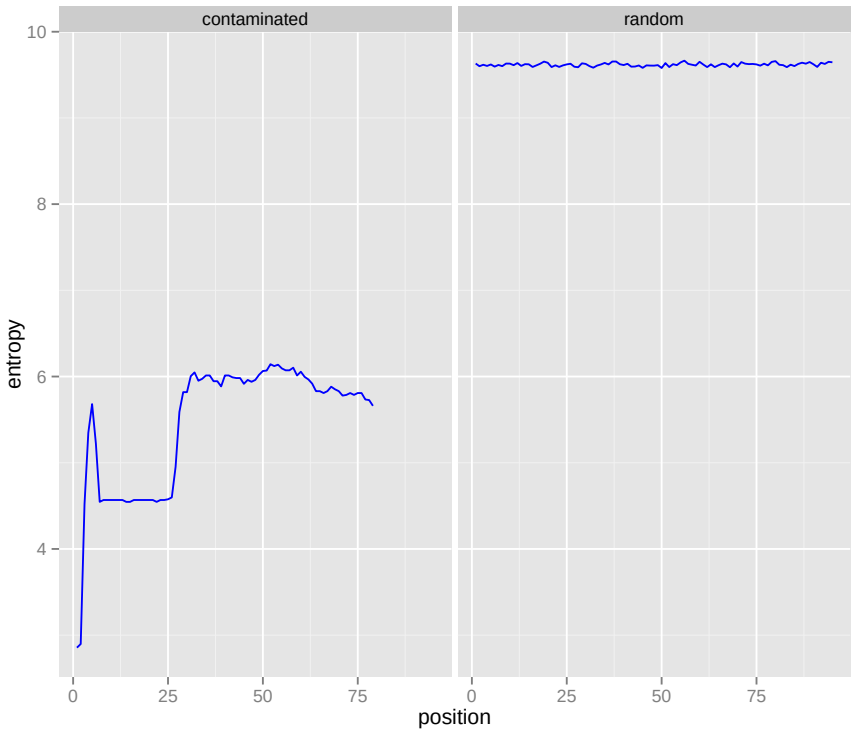

Figure 7: Shannon Entropy for an Illumina file and a FASTA file made from uniformly and randomly sampling bases.

4 k-mer Hashing and Investigating Contamination with Shannon Entropy and K-L Divergence

`readSeqFile` also has an option to hash k-mers. By default, `readSeqFile` randomly selects 10% of the reads and hashes their k-mers with `k=6`. The accessor function `getKmer` will return a data frame of k-mer frequency by position.

The function `kmerEntropyPlot` will plot the Shannon entropy by position, as in Figure 7. `kmerEntropyPlot` will handle any object that inherits from `SequenceSummary` or a named list of such objects.

```
> s.rand <- readSeqFile(system.file('extdata', 'random.fasta', package='qrqc'), type="fasta")
> kmerEntropyPlot(list("contaminated"=s.fastq, "random"=s.rand))
```

To look for possible contamination in this k-mer frequency data, the function `calcKL` calculates the Kullback-Leibler divergence between the distribution of k-mers at a particular positions (for all positions) and the distribution of k-mers across all positions. The result is a data frame of the K-L terms, of which a subset are stacked in the plot produced by `kmerKLPlot`, as in Figure 8.

```
> kmerKLPlot(s.fastq)
```

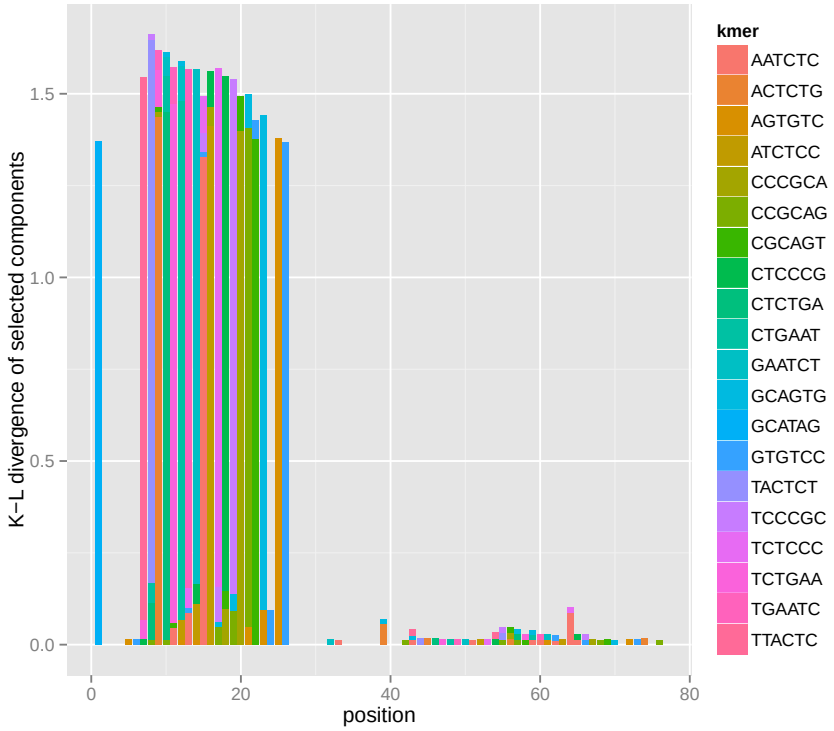

Figure 8: K-L terms for a subset of top k-mers.

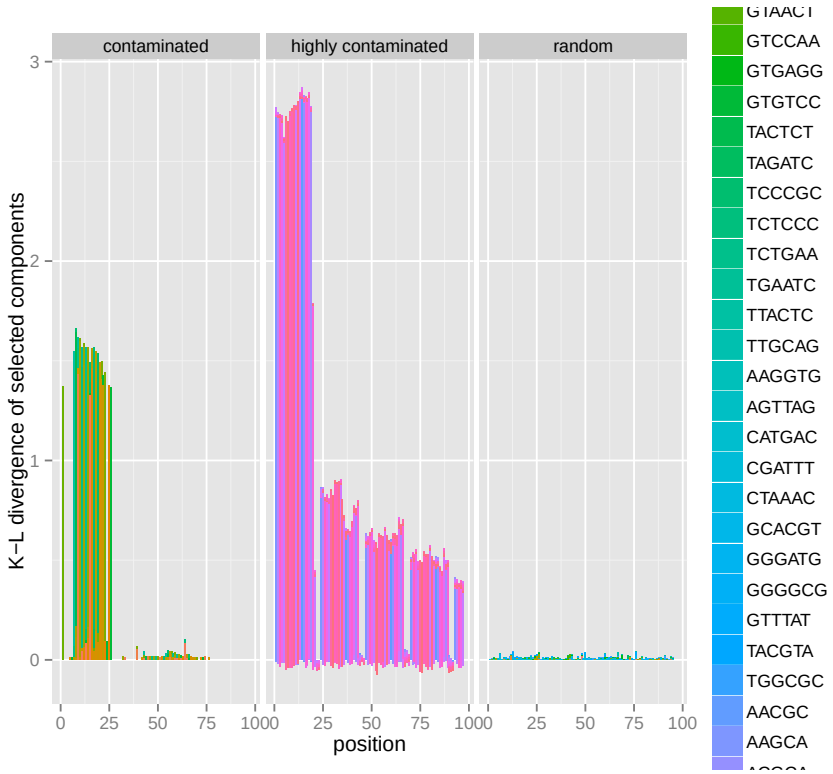

Figure 9: K-L terms for a subset of top k-mers for many `SequenceSummary` objects.

Note that the total height is *proportional* to the actual K-L divergence; because only a subset of the k-mers are used to avoid over plotting, this is not the full K-L divergence (it's a subset of the sample space of k-mers). The number of top k-mers (determined by finding the k-mers with the highest K-L terms) is controlled by setting the number of k-mers to include with `n.kmers`. Passing a list of objects that inherit from `SequenceSummary` will facet the K-L divergence term plot, as in Figure 9

```
> contam.file <- system.file('extdata', 'test-contam.fastq', package='qrqc')
> s.contam <- readSeqFile(contam.file, kmer=TRUE, k=5)
> kmerKLPlot(list("contaminated"=s.fastq, "random"=s.rand,
+   "highly contaminated"=s.contam))
```

5 Customizing Graphics with *ggplot2*

A major motivating factor of using *ggplot2* with *qrqc* is that it allows easy customization of plots without requiring plotting methods to have excessive arguments. For example, if we wish to flip the coordinates of a plot, we easily with `coord_flip()`, as in Figure 10.

```
> basePlot(s.fastq, geom="bar") + coord_flip()
```

`scale_x_continuous` and `scale_y_continuous` can be used to change axis labels and limits of the window. For example, one can focus on only the 3'-end bases, as in Figure 11. In this example, we also use `theme_bw()` to remove *ggplot2*'s default grey background.

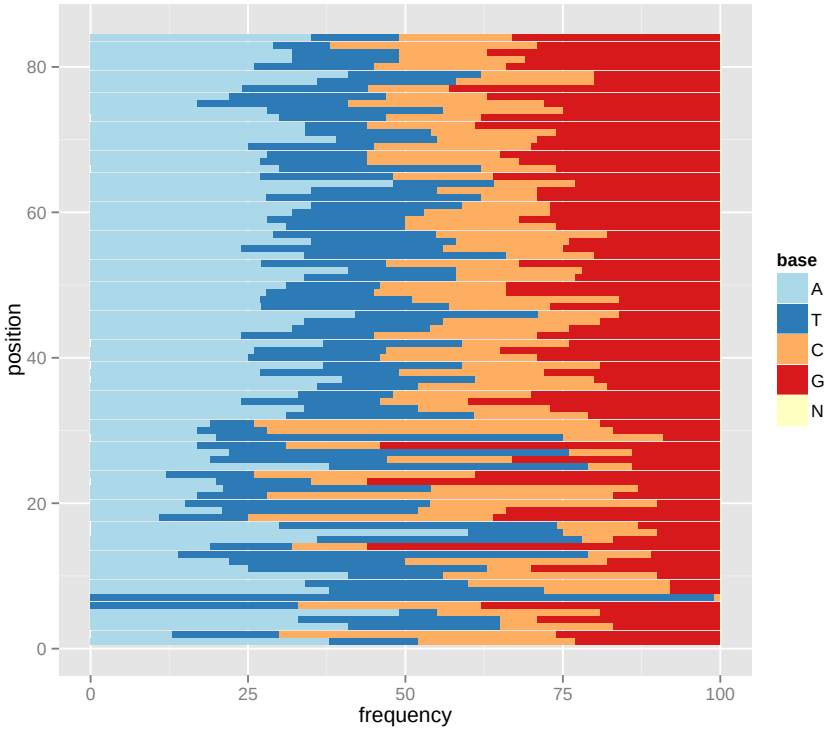

Figure 10: Flipped coordinate base plot.

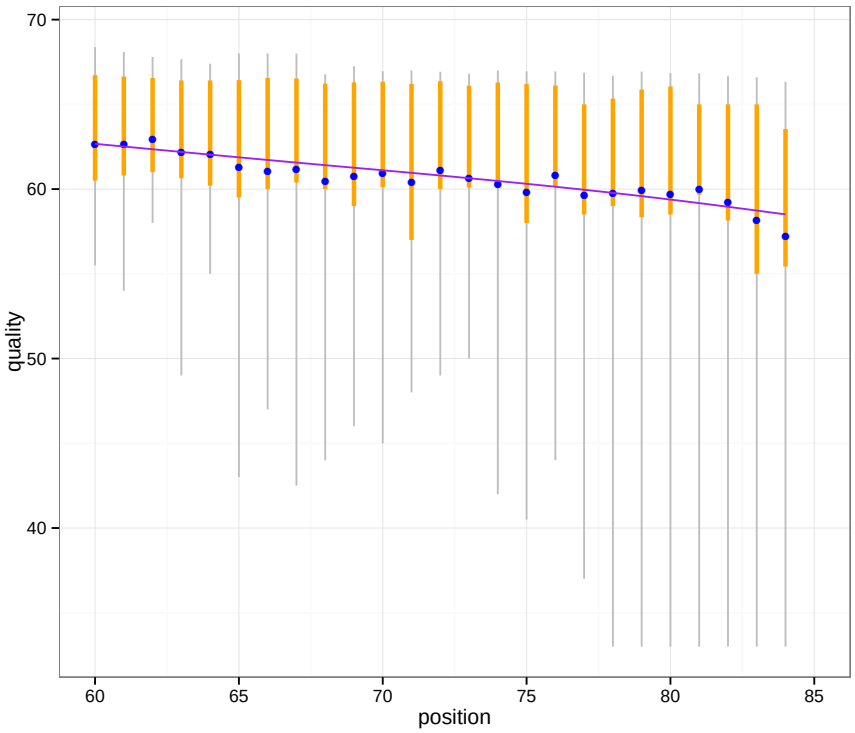

Figure 11: Controlling the x-axis.

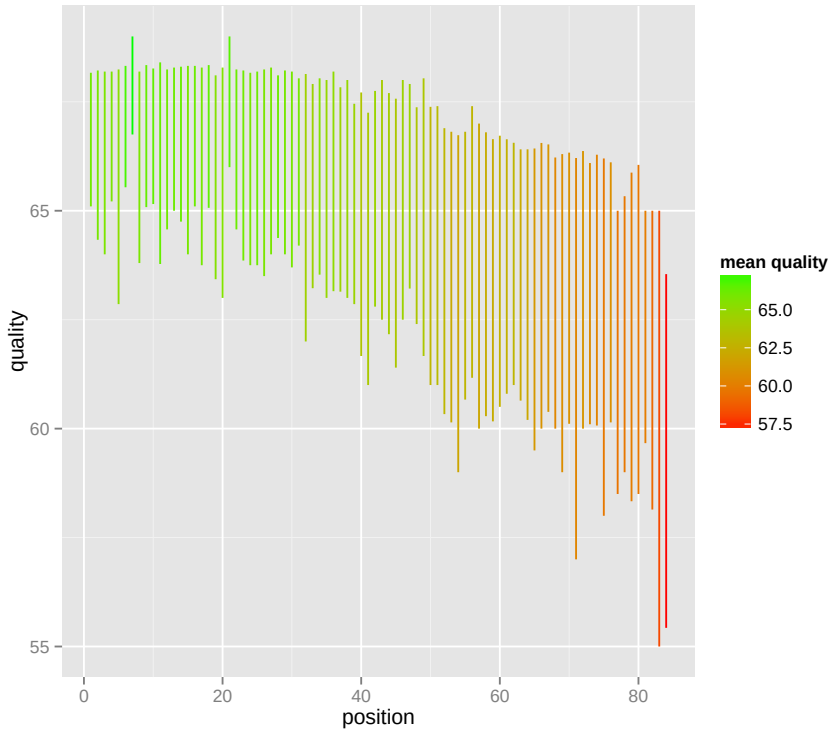

Figure 12: Accessor functions with custom *ggplot2* plots.

```
> qualPlot(s.fastq) + scale_x_continuous(limits=c(60, 85)) + theme_bw()
```

6 Accessor Functions

One can access the summary data gathered by `readSeqFile` through the slots in an object that inherits from `SequenceSummary`. However, these are straight from the C function used by `readSeqFile` and not easy to work with. The accessor functions `getBase`, `getBaseProp`, `getQual`, `getSeqlen`, `getKmer`, and `getMCQual` return data frames that can then be used directly with *ggplot2*.

```
> ggplot(getQual(s.fastq)) + geom_linerange(aes(x=position, ymin=lower,
+       ymax=upper, color=mean)) + scale_color_gradient("mean quality",
+       low="red", high="green") + scale_y_continuous("quality")
```

7 HTML Report Generation

With the help of *brew* and *xtable*, *qrqc* can generate an HTML summary report. This is created with `makeReport`. Reports are stored in their own directories, with images kept under 'images/' and the report is "report.html". A specific output directory can be passed through the `outputDir` argument, but the present directory is used as a default. Multiple reports generated in the same directory will have an incremental naming scheme.

8 Working with the FASTQSummary and FASTASummary classes

qrqc provides the `FASTQSummary` and `FASTASummary` classes for users to build functions and applications around. Both inherit from `SequenceSummary`, which should not be used directly.

`FASTASummary` has the same slots as `FASTQSummary`, except the latter provides additional slots for quality information. Both contain:

- `filename`: the filename of the file read and summarized with `readSeqFile`.
- `base.freqs`: a data frame containing the frequency counts of each base.
- `seq.lengths`: a numeric vector containing the sequences lengths (counts by position).
- `hash`: a numeric vector containing the counts of unique sequences (the actual sequences are the names attribute).
- `hash.propa` numeric value indicating the proportion of sequences that were sampled for hashing.
- `kmera` data frame of k-mer frequency by position.
- `k`an integer value indicating the k-mer size 'k'.
- `hashed`: a logical indicating whether sequence hashing to count unique sequences was done.
- `kmers.hashed`: a logical indicating whether k-mer hashing was done.

Additionally, `FASTQSummary` provides:

- `qual.freqs`: a data frame containing the counts of bases of a particular quality by position.
- `mean.qual`: a numeric giving the mean quality, weighted by sequence lengths.

9 Acknowledgements

Thanks to Simon Andrews for his work on FastQC (a similar program written in Java) from which this project was inspired. Also thanks to Joseph Fass and Dawei Lin for their advice and helpful comments.

Thanks to Heng Li for his work on `kseq.h` `khash.h` (both MIT License) which this package uses through *RSamtools*. More on these header files can be found at `http://lh3lh3.users.sourceforge.net/kseq.shtml` and `http://attractivechaos.awardspace.com/khash.h.html`.

`Sickle` can be downloaded or cloned from the UC Davis Bioinformatics Github repository: `https://github.com/ucdavis-bioinformatics/sickle`.

10 Session Info

```
> sessionInfo()
```

```
R version 3.1.0 (2014-04-10)
Platform: x86_64-apple-darwin13.1.0 (64-bit)

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  stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] mgcv_1.7-29          nlme_3.1-117         qrqc_1.18.0
 [4] testthat_0.8.1       Rsamtools_1.16.0     GenomicRanges_1.16.3
 [7] GenomeInfoDb_1.0.2   xtable_1.7-3         brew_1.0-6
[10] biovizBase_1.12.1    Biostrings_2.32.0    XVector_0.4.0
[13] IRanges_1.22.5       BiocGenerics_0.10.0  ggplot2_0.9.3.1
[16] reshape_0.8.5

loaded via a namespace (and not attached):
 [1] AnnotationDbi_1.26.0     BBmisc_1.6               BSgenome_1.32.0
 [4] BatchJobs_1.2            Biobase_2.24.0           BiocParallel_0.6.0
 [7] DBI_0.2-7                Formula_1.1-1            GenomicAlignments_1.0.1
[10] GenomicFeatures_1.16.0   Hmisc_3.14-4             MASS_7.3-32
[13] Matrix_1.1-3             RColorBrewer_1.0-5       RCurl_1.95-4.1
[16] RSQLite_0.11.4           Rcpp_0.11.1              VariantAnnotation_1.10.0
[19] XML_3.98-1.1             biomaRt_2.20.0           bitops_1.0-6
[22] cluster_1.15.2           codetools_0.2-8          colorspace_1.2-4
[25] dichromat_2.0-0          digest_0.6.4             fail_1.2
[28] foreach_1.4.2            grid_3.1.0               gtable_0.1.2
[31] iterators_1.0.7          labeling_0.2             lattice_0.20-29
[34] latticeExtra_0.6-26      munsell_0.4.2            plyr_1.8.1
[37] proto_0.3-10             reshape2_1.4             rtracklayer_1.24.0
[40] scales_0.2.4             sendmailR_1.1-2          splines_3.1.0
[43] stats4_3.1.0             stringr_0.6.2            survival_2.37-7
[46] tools_3.1.0              zlibbioc_1.10.0
```