Bioinformatics, 33(20), 2017, 3311–3313
doi: 10.1093/bioinformatics/btx450
Advance Access Publication Date: 14 July 2017
Applications Note

OXFORD

Genetics and population analysis

VariantTools: an extensible framework for developing and testing variant callers

Michael Lawrence[1,*] and Robert Gentleman[2]

[1]Department of Bioinformatics, Genentech, South San Francisco, CA, USA and [2]23andMe, Mountain View, CA, USA

*To whom correspondence should be addressed.

Associate Editor: Oliver Stegle

Received on September 2, 2016; revised on June 3, 2017; editorial decision on July 9, 2017; accepted on July 11, 2017

Abstract

Motivation: Variant calling is the complex task of separating real polymorphisms from errors. The appropriate strategy will depend on characteristics of the sample, the sequencing methodology and on the questions of interest.

Results: We present VariantTools, an extensible framework for developing and testing variant callers. There are facilities for reproducibly tallying, filtering, flagging and annotating variants. The tools are extensible, modular and flexible, so that they are tunable to particular use cases, and they interoperate with existing analysis software so that they can be embedded in established work flows.

Availability and implementation: VariantTools is available from http://www.bioconductor.org/.

Contact: michafla@gene.com

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Variant calling is not an exact science, and we often need to diagnose variant caller output, or implement a custom variant caller when assumptions are violated (Reumers *et al.*, 2012). There are many contexts in which methods are still evolving, such as RNA editing, structural variants and alignments to graph-based genomes. Caller diagnostics assist decision making during the design and maintenance of variant calling pipelines by elucidating the effect of filtering parameters and algorithmic changes on the output.

Filtering is fundamental to generating, annotating and diagnosing variant calls. The appropriateness of a filter depends on the dataset and the question. Therefore, we need a flexible and extensible filtering framework that enables code reuse, reproducibility and strategic decision making, where filters operate on standard data structures and storage formats, so as to enable integration with existing workflows and tracking of data provenance.

Filter diagnostics require annotating the data with filter results, without actually removing any data. An example is the quality tranche annotation generated by the GATK pipeline (Auwera *et al.*, 2013). Conceptually the result is a matrix with rows corresponding to variants and columns to filters. A true value in a cell indicates that a variant passes a filter. Using a matrix of filter results, we analyze the outcome as if the filters were applied in series, or in parallel, i.e. independently, in order to determine which filters are discarding (or not) a particular variant, and which variants a particular filter is excluding. We call filter-based annotation *soft* filtering, while *hard* filtering refers to the removal of variants that fail a filter.

The VariantTools package extends Bioconductor (Huber *et al.*, 2015) with a host of filters for exploring variant calls and developing variant calling pipelines.

2 Results

Filters operate on the standard Bioconductor *VRanges* data structure. *VRanges* extends the generic genomic data container *GRanges* (Lawrence *et al.*, 2013) to formally represent variant calls, including sequence depth information, filter results and filter provenance. The user can import and export VCF files as *VRanges* objects. The gmapR package (Barr *et al.*, 2016) computes tallies from BAM files and returns them as *VRanges* objects.

Built-in filters consider minimum coverage, minimum alt count, minimum alt frequency, strand bias and read position bias. There are filters for identifying clumping of variants along the chromosome, as well as for determining whether a variant is unique to a sample within

a pair (e.g. tumor-specific variants). VariantTools contains a diploid genotyper and an algorithm that decides whether a variant is callable, based on coverage and a power calculation. The filters and algorithms are meant for exploratory and diagnostic purposes, so we designed them to be simple, efficient and easy to interpret.

To demonstrate the capabilities of VariantTools, we load some precomputed nucleotide tallies from the VariantToolsData package (Lawrence, 2016). The code for this example is available in the Supplementary Material. We computed the tallies from the sequencing of a 50/50 mixture of the HapMap cell lines NA12878 and NA19240. We biochemically mixed the samples in triplicate and now sum the read counts over the replicates to maximize effective coverage. For convenience, the data have been subset to positions with at least one non-reference read and within a 1 Mb region around the TP53 gene.

```
> library(VariantToolsData)
> tallies <- sumDepths(get(data(tallies)))
```

VariantTools provides filters in sets represented by list-like *FilterRules* objects, a standard Bioconductor data structure. Users can construct filters one at a time and then combine them into *ad hoc* pipelines. For quality filtering, the packge includes filters based on a Fisher test of strand bias and the median distance from nearest end (MDFNE), a measure of read position imbalance.

```
> library(VariantTools)
> tallies <- softFilter(tallies, VariantQAFilters())
```

The call to softFilter() applies the filters, returning the status of all variants (no removal). We can compute statistics on the filter results:

```
> summary(softFilterMatrix(tallies), percent=TRUE)
  <initial>    mdfne    fisherStrand    <final>
      1.000    0.758           0.011      0.010
```

We find that the MDFNE filter passed only about 76% of the positions, while the Fisher's Exact Test of strand bias passed all of them independently. Strand bias does not appear to be an issue but perhaps mapping artifacts are leading to a position bias, shrinking the median distance to the nearest end of the read.

The package includes filters for winnowing noisy tallies down to a working set of variant calls. They are based on the count and frequency of reads supporting each variant. The callVariants() function applies the default set of calling filters in hard fashion, restricting the dataset, where only the first filter sees all of the data.

```
> variants <- callVariants(tallies)
```

The variants object preserves which filters were applied, so by suppling the original input, we can count how many variants were kept after each filter, applied in series:

```
> summary(hardFilters(variants), tallies, serial=TRUE)
     <initial>       nonRef        nonNRef
         75060        75060          75060
     readCount likelihoodRatio    <final>
         8720           8115          8115
```

From these statistics, the user might decide to tweak a cutoff value, or drop a filter entirely. They also inform on trends in the data that will affect other variant calling tasks. Here, we find that the requirement of 2 or more alt reads is excluding the majority of the calls.

The *FilterRules* framework is extensible, so users can define custom filters to interrogate aspects that are particularly relevant to their data. The user defines a filter with an ordinary R function, which can consider any attribute of the data. For example, we can identify those variants that overlap a homopolymer:

```
> HomopolymerFilter <-
+     function(genome, which, maxlen) {
+         seq <- getSeq(genome, which)
+         rle <- RleList(lapply(seq, as.raw),
+
compress=FALSE)
+         long <- runLength(rle) > maxlen
+         hp <- as(ranges(rle)[long], "GRanges")
+         function(x) {
+                             ! (x
+         }
+     }
> which <- VariantToolsData::TP53Region()
> library(BSgenome.Hsapiens.UCSC.hg19)
> genome <- BSgenome.Hsapiens.UCSC.hg19
> hp <- HomopolymerFilter(genome, which, 8)
> filters <- FilterRules(list(homopolymer=hp))
> summary(filters, tallies, percent=TRUE)
  <initial>    homopolymer    <final>
      1.000          0.995      0.995
```

We could easily experiment with different homopolymer length cutoffs.

We compare our NA12878 variants after 50% dilution with those of Zook *et al.* (2015):

```
> table(raw=giab
+             called=giab
         called
raw      FALSE    TRUE
  FALSE    16       0
  TRUE      0    2080
```

Our tallies object has 2080 variants also in the Zook data (giab), and there are 16 variants that are not represented in our tallies.

3 Discussion

We provide a tool for experimenting with different filtering methods on variant calls, quickly prototyping ideas and assessing the effects of different methods. For example, we have applied it to an evaluation of the effect of variant frequency on caller performance (Lawrence *et al.*, 2015).

Filtering tasks rely heavily on vectorized computations, which R performs efficiently, as well as specialized algorithms implemented in native code. For example, computing the homopolymer overlap filter for the NA12878 variants (1.35 million HPs, 3.16 million variants) takes less than half a second on a modern Macbook Pro laptop. When combined with existing parallel computing functionality in R, simple filter implementations are scalable. For large scale projects, on the order of thousands of individuals or more, it may be worth producing optimized special purpose implementations once

the prototypes have been explored and a final filtering scheme decided on.

4 Conclusion

VariantTools provides a tool box for exploring different filtering strategies. It integrates with existing Bioconductor workflows via standard data structures. There remain many problems in variant identification where an appropriate filtering strategy is not known and in those cases VariantTools provides a low cost method for exploring realistic sized datasets.

Acknowledgement

We acknowledge Jeremiah Degenhardt for his work on the prototype that evolved into VariantTools.

Conflict of Interest: none declared.

References

Auwera,G.A. *et al*. (2013) From fastq data to high-confidence variant calls: the genome analysis toolkit best practices pipeline. *Curr. Protoc. Bioinf*., **43**, 11–10.

Barr,C. *et al*. (2016) *gmapR: An R interface to the GMAP/GSNAP/GSTRUCT suite*. R package version 1.17.1.

Huber,W. *et al*. (2015) Orchestrating high-throughput genomic analysis with Bioconductor. *Nat. Methods*, **12**, 115–121.

Lawrence,M. (2016) *VariantToolsData: Data for the VariantTools tutorial*. R package version 0.99.0.

Lawrence,M. *et al*. (2013) Software for computing and annotating genomic ranges. *PLoS Comput. Biol*., **9**.

Lawrence,M. *et al*. (2015) Genomic variant calling: flexible tools and a diagnostic data set. *bioRxiv*.

Reumers,J. *et al*. (2012) Optimized filtering reduces the error rate in detecting genomic variants by short-read sequencing. *Nat. Biotechnol*., **30**, 61–68.

Zook,J.M. *et al*. (2015) Extensive sequencing of seven human genomes to characterize benchmark reference materials. *bioRxiv*.