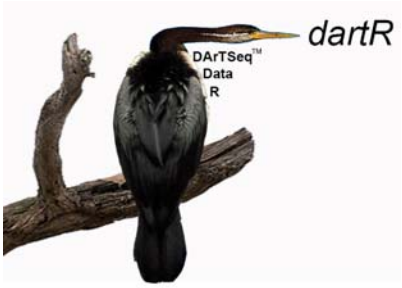
Wildlife Genomics@UC
wildlifegenomics.biomatix.org

UNIVERSITY OF CANBERRA

SNP Analysis using dartR



Guide to Basic Filtering

Version 2

Copies of the latest version of this tutorial are available from:

The Institute for Applied Ecology
University of Canberra ACT 2601
Australia

Email: georges@aerg.canberra.edu.au

dartR is a collaboration between the University of Canberra, CSIRO and Diversity Arrays Technology, and is supported with funding from the ACT Priority Investment Program, CSIRO and the University of Canberra.

Contents

Session 1: Basic Filtering

Overview

Calling SNPs in genotyping by sequencing is not an exact science. Judgements need to be made at various points in the workflow to increase the likelihood of a correct call at a particular locus. DArT Pty Ltd has already done much of the filtering of the sequences used to generate your SNPs that would normally be undertaken by researchers who generate their own ddRAD data. Here we present some other filters that you might wish to apply to increase the reliability of the data retained in your SNP or SilicoDArT dataset.

For each filter parameter offered by dartR, we provide both a report function and a filtering function (e.g. `gl.report.reproducibility` us associated with `gl.filter.reproducability`). The reporting function provides descriptive statistics for the parameter of choice for your data. Inspection of these outputs will assist you to identify appropriate filter thresholds. Hence, as a standard workflow, it is a good idea to run the `gl.report` functions in advance of applying each `gl.filter` function to provide a foundation for selecting thresholds.

Several filters are available to improve the quality of the data represented in your genlight object. Some of these are specific to dartR data (e.g. `gl.filter.reproducability`), but most will work on any data provided they are in the appropriate genlight format.

The basic ones are:

`gl <- gl.filter.reproducibility()`	filter out loci for which the reproducibility (strictly repeatability) is less than a specified threshold, say threshold = 0.99
`gl <- gl.filter.callrate()`	filter out loci or individuals for which the call rate (rate of non-missing values) is less than a specified threshold, say threshold = 0.95
`gl <- gl.filter.monomorphs()`	filter out monomorphic loci and loci that are scored all NA
`gl <- gl.filter.allna`	filter out loci that are all missing values (NA)
`gl <-gl.filter.secondaries()`	filter out SNPs that share a sequence tag, except one retained at random
`gl <- gl.filter.hamming()`	filter out loci that differ from each other by less than a specified number of base pairs
`gl <- gl.filter.rdepth()`	filter out loci with exceptionally low or high read depth (coverage)
`gl <- gl.filter.taglength()`	filter out loci for which the tag length is less that a threshold

`gl <- gl.filter.overshoot()` filter out loci where the SNP location lies outside the trimmed sequence tag

The order of filtering can be important and requires some thought. Filtering on call rate by individual before filtering on call rate by locus or choosing the alternative order will depend on the weight placed on losing individuals versus losing loci, for example.

Example: Filtering on reproducibility

First, we should examine the distribution of reproducibility measures (RepAvg) in our dataset.

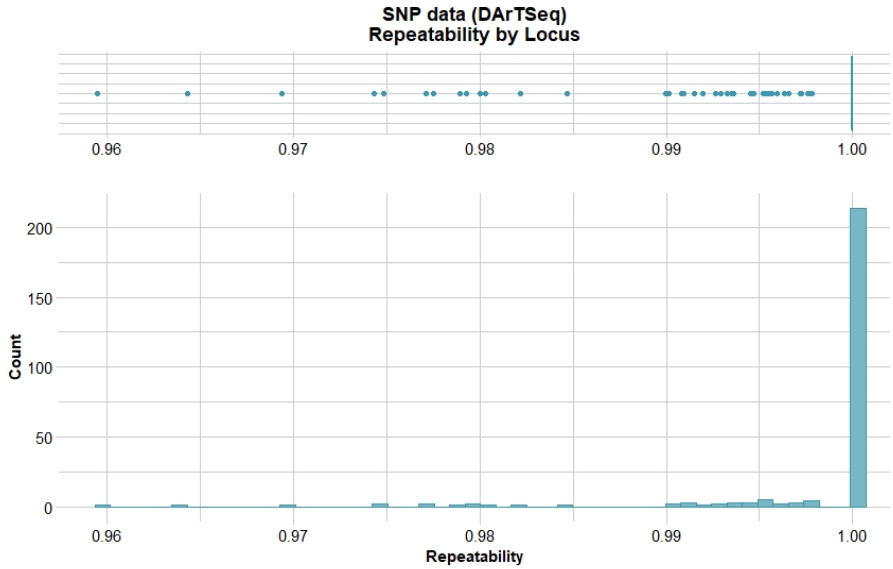
```
gl.report.reproducibility(testset.gl,plot=TRUE)
```

```
Starting gl.report.reproducibility
Processing SNP data
Reporting Repeatability by Locus
No. of loci = 255
No. of individuals = 250
Minimum      :  0.959459
1st quartile :  1
Median       :  1
Mean         :  0.9981525
3r quartile  :  1
Maximum      :  1
Missing Rate Overall:  0.12

Quantile Threshold Retained Percent Filtered Percent
1      100% 1.0000000      214    83.9       41    16.1
2       95% 1.0000000      214    83.9       41    16.1
3       90% 1.0000000      214    83.9       41    16.1
4       85% 1.0000000      214    83.9       41    16.1
5       80% 1.0000000      214    83.9       41    16.1
6       75% 1.0000000      214    83.9       41    16.1
7       70% 1.0000000      214    83.9       41    16.1
8       65% 1.0000000      214    83.9       41    16.1
9       60% 1.0000000      214    83.9       41    16.1
10      55% 1.0000000      214    83.9       41    16.1
11      50% 1.0000000      214    83.9       41    16.1
12      45% 1.0000000      214    83.9       41    16.1
13      40% 1.0000000      214    83.9       41    16.1
14      35% 1.0000000      214    83.9       41    16.1
15      30% 1.0000000      214    83.9       41    16.1
16      25% 1.0000000      214    83.9       41    16.1
17      20% 1.0000000      214    83.9       41    16.1
18      15% 0.9976873      216    84.7       39    15.3
19      10% 0.9945940      229    89.8       26    10.2
20       5% 0.9883732      242    94.9       13     5.1
21       0% 0.9594590      255   100.0        0     0.0

Completed: gl.report.reproducibility
```

This output is useful in that it provides an indication of how much data will be lost, when filtering, for each choice of a threshold value. Clearly, with a minimum repeatability of 0.96 and a maximum of 1 across loci, we can be fairly stringent in our choice of a threshold. A value of 0.99 will not result in the loss of much data (16.1%). The judgement can also be made on the basis of the graphical output.



We now filter on selecting a threshold value of 0.99.

```
gl <- gl.filter.reproducibility( testset.gl,
         threshold=0.99, verbose = 3)
```

```
Starting gl.filter.reproducibility
  Processing a SNP dataset
  Identifying loci with repeatability below : 0.99
  Removing loci with repeatability less than 0.99

  Summary of filtered dataset
  Retaining loci with repeatability >= 0.99
  Original no. of loci: 255
  No. of loci discarded: 14
  No. of loci retained: 241
  No. of individuals: 250
  No. of populations:  30

Completed: gl.filter.reproducibility
```

Only 14 loci out of 255 were deleted.

It is wise not to filter too heavily on reproducibility. A threshold of 1 is often tempting but can result is some bias being introduced.

Exercises

1. Just in case you have accidentally modified the genlight object `gl`, recreate it by copying it from `testset.gl`.

   ```
   gl <- testset.gl
   ```

2. Filter `gl` using the filters listed above. Request a report first to inform your choice of threshold. Be sure to set plot=TRUE to examine the distribution of each parameter and optionally smearplot=TRUE to examine the smear plot.

Recalculating locus metadata after filtering

Remember, the locus metrics are no longer valid if individuals or populations are deleted from the dataset. For example, if you filter out a population for which the individuals have particularly bad call rates, then the call rate parameter held in the locus metrics will no longer be accurate. It will need to be recalculated. This is true of many of the locus metrics.

So, after filtering your data, it is wise to recalculate the locus metrics with

```
gl <- gl.recalc.metrics(gl)
```

Try this for yourself on genlight object `gl` after filtering or on your own data

Similarly, when filtering has resulted in removal of some individuals or populations, variation at several loci may be lost. Some loci may even be scored as missing across all individuals. You may wish to remove these monomorphic loci from your dataset with

```
gl <- gl.filter.monomorphs(gl)
```

Try this for yourself on genlight object `gl` after filtering or on your own data

Filtering order

It is often unclear as to what order the filtering steps should take in your workflow. Does one filter on call rate by individual first then call rate by locus or the other way around. There is no correct answer to this, as it depends on whether you value retaining loci over retaining individuals. Usually, you would not want to lose individuals from your dataset, so filtering out those loci will low call rates would come first, filtering on individuals second, though this is not always the case. A starting point for considering a workflow might be:

1. Optionally filter on Hamming Distance if the DArT threshold is considered too lenient.
2. Filter out secondaries (all but one SNP retained per sequence tag)
3. Optionally filter on read depth if the DArT threshold is considered too lenient.
4. Filter out loci with a reproducibility below a particular threshold (say 0.98)
5. Filter out loci with a call rate below a particular threshold (say 0.95)
6. Filter out individuals with a call rate below a particular threshold (say 0.80)

Be careful not to over-filter. The objective is to get an appropriate balance between signal to noise ratio, not to eliminate noise altogether at the expense of also taking out some signal. This balance will depend on downstream application.

An example of a filtering sequence might be

```
gl <-gl.filter.secondaries(gl)
gl <- gl.filter.rdepth(gl)
gl <- gl.filter.reproducibility(gl)
gl <- gl.filter.callrate(gl, method="loc")
gl <- gl.filter.callrate(gl, method="ind")
gl <- gl.filter.monomorphs(gl)
```

Where have we come?

The above Session was designed to give you an overview of the scripts in dartR for filtering your data. Having completed this Session, you should now able to:

- Filter on call rate, repeatability, secondaries, hamming distance, and minor allele frequency.
- Recalculate locus metrics after deleting individuals or populations as part of the filtering process.
- Filter out resultant monomorphic loci.

Having played with the various filters, you should also have a good appreciation of how to select appropriate thresholds for filtering and how to introduce a sequence of filtering steps into your workflows.

Further reading

Jombart T. and Caitlin Collins, C. (2015). Analysing genome-wide SNP data using adegenet 2.0.0. http://adegenet.r-forge.r-project.org/files/tutorial-genomics.pdf

Gruber, B., Unmack, P.J., Berry, O. and Georges, A. 2018. dartR: an R package to facilitate analysis of SNP data generated from reduced representation genome sequencing. Molecular Ecology Resources, 18:691–699

Ende