CGHcall: Calling aberrations for array CGH tumor profiles.

Sjoerd Vosse and Mark van de Wiel

October 17, 2016

Department of Epidemiology & Biostatistics
VU University Medical Center

mark.vdwiel@vumc.nl

Contents

1 Overview

CGHcall allows users to make an objective and effective classification of their aCGH data into copy number states (loss, normal, gain or amplification). This document provides an overview on the usage of the CGHcall package. For more detailed information on the algorithm and assumptions we refer to the article (**?**) and its supplementary material. As example data we attached the first five samples of the Wilting dataset (**?**). After filtering and selecting only the autosomal 4709 datapoints remained.

2 Example

In this section we will use CGHcall to call and visualize the aberrations in the dataset described above. First, we load the package and the data:

```
> library(CGHcall)
> data(Wilting)
> Wilting <- make_cghRaw(Wilting)
```

Next, we apply the `preprocess` function which:

- removes data with unknown or invalid position information.
- shrinks the data to `nchrom` chromosomes.
- removes data with more than `maxmiss` % missing values.
- imputes missing values using `impute.knn` from the package `impute` (**?**).

```
> cghdata <- preprocess(Wilting, maxmiss=30, nchrom=22)
```

```
Changing impute.knn parameter k from 10 to 4 due to small sample size.
```

To be able to compare profiles they need to be normalized. In this package we first provide very basic global median or mode normalization. This function also contains smoothing of outliers as implemented in the DNAcopy package (**?**). Furthermore, when the proportion of tumor cells is not 100% the ratios can be corrected. See the article and the supplementary material for more information on cellularity correction (**?**).

```
> norm.cghdata <- normalize(cghdata, method="median", smoothOutliers=TRUE)
```

```
Applying median normalization ...
Smoothing outliers ...
```

The next step is segmentation of the data. This package only provides a wrapper function that applies the `DNAcopy` algorithm (**?**). It provides extra functionality by allowing to undo splits differently for long and short segments, respectively. In the example below short segments are smaller than clen=10 probes, and for such segments undo.splits is effective when segments are less than undo.SD=3 (sd) apart. For long segments a less stringent criterion holds: undo when less than undo.SD/relSDlong = 3/5 (sd) apart. If, for two consecutive segements, one is short and one is long, splits are undone in the same way as for two consecutive short segments. To save time we will limit our analysis to the first two samples from here on.

```
> norm.cghdata <- norm.cghdata[,1:2]
> seg.cghdata <- segmentData(norm.cghdata, method="DNAcopy",undo.splits="sdundo",undo
+ clen=10, relSDlong=5)
```

```
Start data segmentation ..
Analyzing: Sample.1
Analyzing: Sample.2
```

Post-segmentation normalization allows to better set the zero level after segmentation.

```
> postseg.cghdata <- postsegnormalize(seg.cghdata)
```

Now that the data have been normalized and segments have been defined, we need to determine which segments should be classified as double losses, losses, normal, gains or amplifications. Cellularity correction is now provided WITHIN the calling step (as opposed to some earlier of CGHcall)

```
> tumor.prop <- c(0.75, 0.9)
> result <- CGHcall(postseg.cghdata,nclass=5,cellularity=tumor.prop)

[1] "Total number of segments present in the data: 90"
[1] "Number of segments used for fitting the model: 90"
```

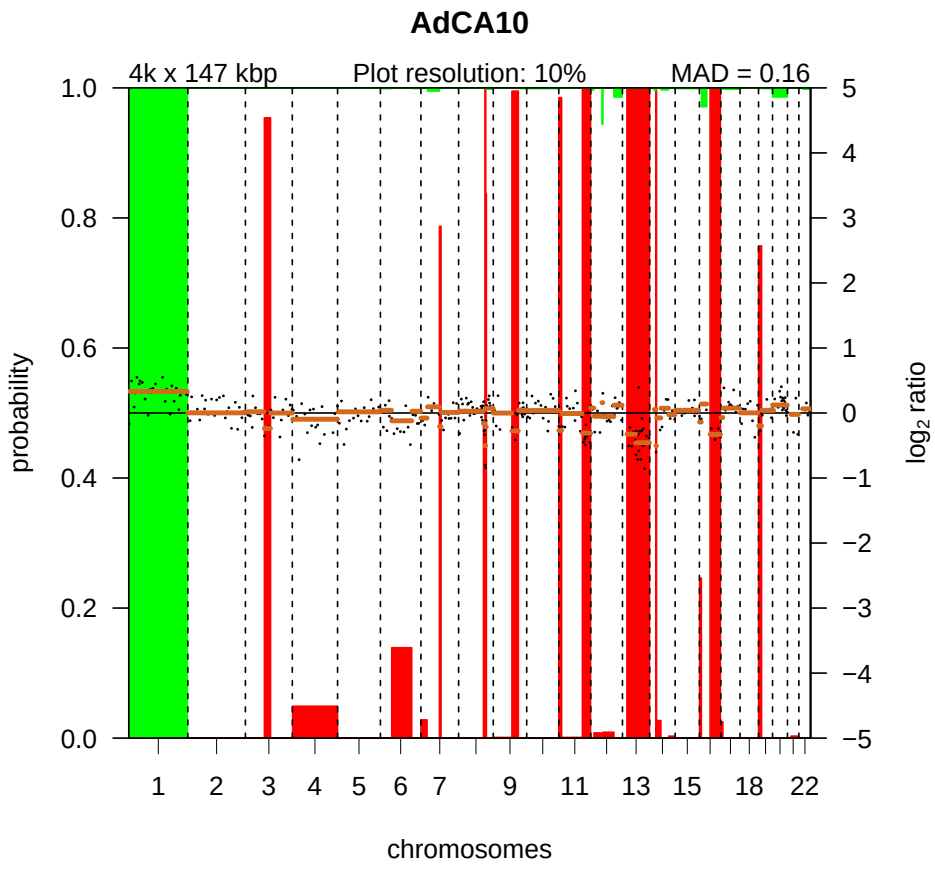
The result of CGHcall needs to be converted to a call object. This can be a large object for large arrays.

```
> result <- ExpandCGHcall(result,postseg.cghdata)
```

To visualize the results per profile we use the `plotProfile` function:

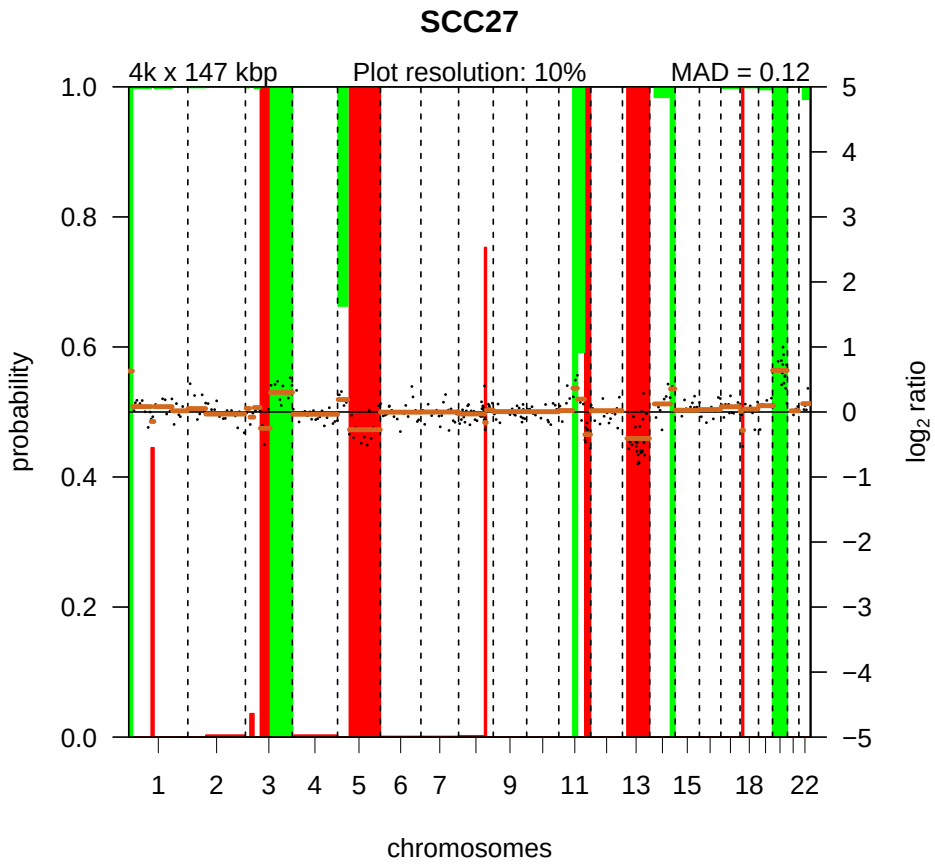
```
> plot(result[,1])

Plotting sample AdCA10
```



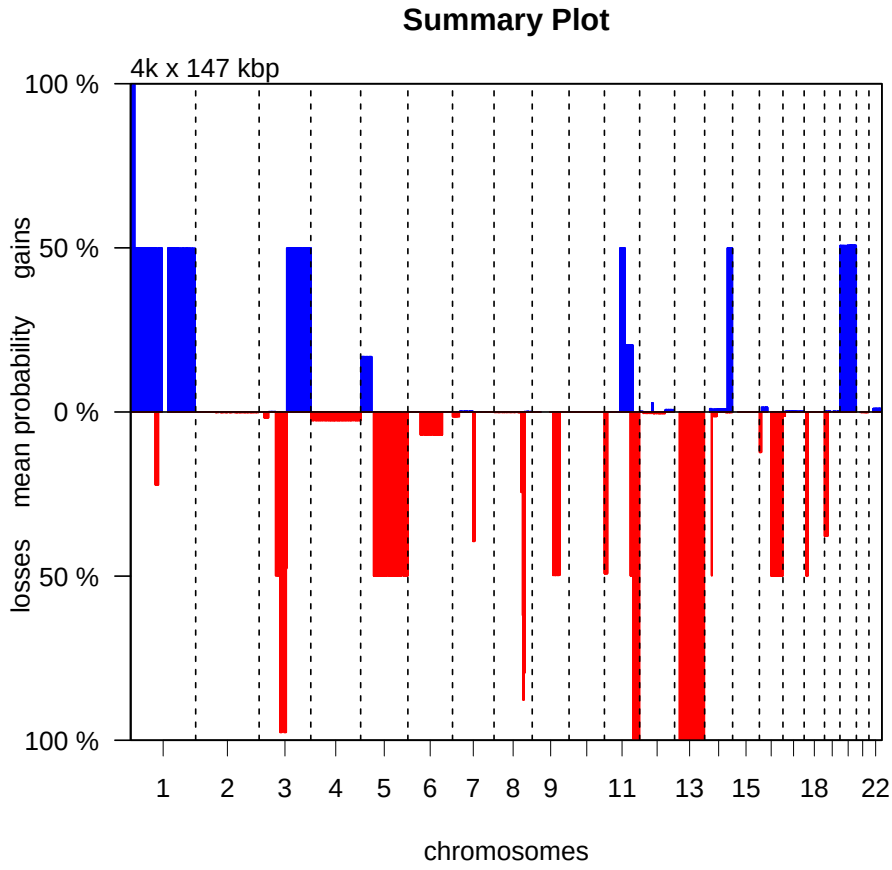
```
> plot(result[,2])

Plotting sample SCC27
```



Alternatively, we can create a summary plot of all the samples:

```
> summaryPlot(result)
```



Or a frequency plot::

```
> frequencyPlotCalls(result)
```