Overview of CNPBayes package

Jacob Carey, Steven Cristiano, and Robert Scharpf

April 30, 2018

Contents

1 Introduction

CNPBayes models multi-modal densities via a hierarchical Bayesian Gaussian mixture model. The major application of this model is the estimation of copy number at copy number polymorphic loci (CNPs). Four versions of the mixture model are implemented: a *standard* model, referred to as a *SingleBatch* (SB) model , that has one mean and standard deviation for each component; a *SingleBatchPooled* (SBP) model that has a pooled estimate of the standard deviation across all mixture components; a *MultiBatch* (MB) model with batch-specific means and standard deviations; and a *MultiBatchPooled* (MBP) model with batch-specific standard deviations that are pooled across mixture components within each batch. For all versions, approximation of the posterior is by Markov Chain Monte Carlo (MCMC) written in C++ using the Rcpp package (Eddelbuettel and François 2011).

For an EM-implementation of Gaussian mixture models for CNPs, see the Bioconductor package CNVtools (Barnes et al. 2008). A Bayesian extension of this model by some of the same authors was developed to automate the analysis of the Welcome Trust Case Control Consortium (WTCCC) genotype data (Cardin et al. 2011) and implemented in the R package CNVCALL (http://niallcardin.com/CNVCALL).

This vignette provides a concise workflow for fitting mixture models in large array-based genome-wide association studies. Other vignettes in this package provide details on the implementation and functions for evaluating convergence.

```
suppressPackageStartupMessages(library(CNPBayes))
suppressPackageStartupMessages(library(SummarizedExperiment))
library(ggplot2)
```

2 Workflow

2.1 Summarizing data at CNPs

Our starting point is a `SummarizedExperiment` containing log R ratios from a SNP array and a `GRangesList` containing deletions and duplications identified from a hidden Markov model.

```
se <- readRDS(system.file("extdata", "simulated_se.rds", package="CNPBayes"))
grl <- readRDS(system.file("extdata", "grl_deletions.rds", package="CNPBayes"))
```

Using this data, we identify the genomic coordinates of the CNP locus and use the median to summarize the log R ratios for each sample. We advise against using the first principal component as a summary statistic in large studies as this may capture batch effects. See Identifying Copy Number Polymorphisms for instructions on finding CNPs with a `SnpArrayExperiment` and `GRangesList`.

```
cnv.region <- consensusCNP(grl, max.width=5e6)
i <- subjectHits(findOverlaps(cnv.region, rowRanges(se)))
med.summary <- matrixStats::colMedians(assays(se)[["cn"]][i, ], na.rm=TRUE)
```

At loci where copy number variants are common, the distribution of the median log R ratios computed above can be approximated by a finite mixture of normal distributions.

2.2 The `gibbs` function

This package provides several implementations of a Bayesian hierarchical normal mixture model, where the implementations differ by whether the component means and variances are batch-specific (multi-batch 'MB' model) or a single batch ('SB' model). In addition, we allow the variance estimates to be pooled across all components within the multi-batch ('MBP') and single-batch ('SBP') models. For large studies involving SNP microarrays, the `combinePlates` function can be useful to identify groups of plates processed in the same batch. As there are only 35 samples in this toy example, we assume that these samples were all processed in the same batch and we will only consider the SB and SBP models. The following code chunk initializes an MCMC parameters object that contain parameters governing the number of burnin iterations, the number of iterations following burnin, how often two save simulated values from the chain (*thin*), and the number of independently initialized models (*nStarts*).

```
mp <- McmcParams(nStarts=5, burnin=500, iter=1000, thin=1)
```

The workhorse function in this package is `gibbs`. This function allows the user to specify one or all of the four possible mixture model types (SB, MB, SBP, MBP) and for each type fit models with mixture components specified by the argument `k_range`. Below, we run 4 chains for 1000 iterations (100 iterations burnin) for only the k=3 SB model. (In practice, we would fit multiple models and specify a range for k. For example, `k_range=c(1, 5)`). As described in the convergence vignette, an attempt is made by `gibbs` to adapt the thinning and burnin parameters if there are indications that the chains have not converged. The models evaluated by `gibbs` are returned in a list where all chains have been combined and the models are sorted by decreasing value of the marginal likelihood. In order to properly assess convergence, this function requires that one run at least 2 independent chains. Below, we specify a `max_burnin` to a small number (400) to speed up computation. A much longer burnin may be required for convergence.

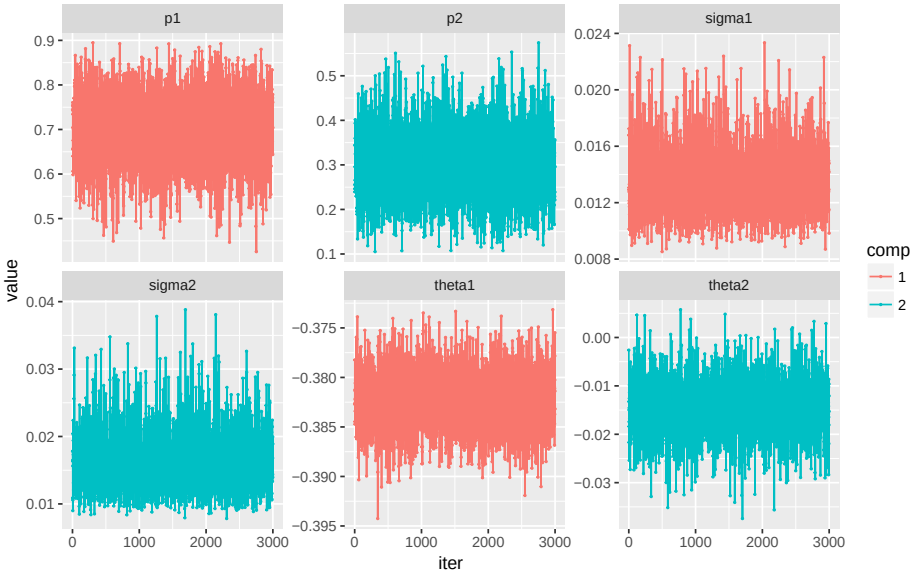
```
model.list <- gibbs(model="SB", dat=med.summary, k_range=c(2, 3), mp=mp,
                    max_burnin=400, top=2)
## Fitting SB models
##   k: 2, burnin: 500, thin: 1
##      r: 1
##      eff size (minimum): 1421
##      eff size (median): 2989.9
##      marginal likelihood: 58.59
##   k: 3, burnin: 500, thin: 1
##      r: 14.43
##      eff size (minimum): 343
##      eff size (median): 1243
```

2.3 Evaluating convergence and goodness of fit

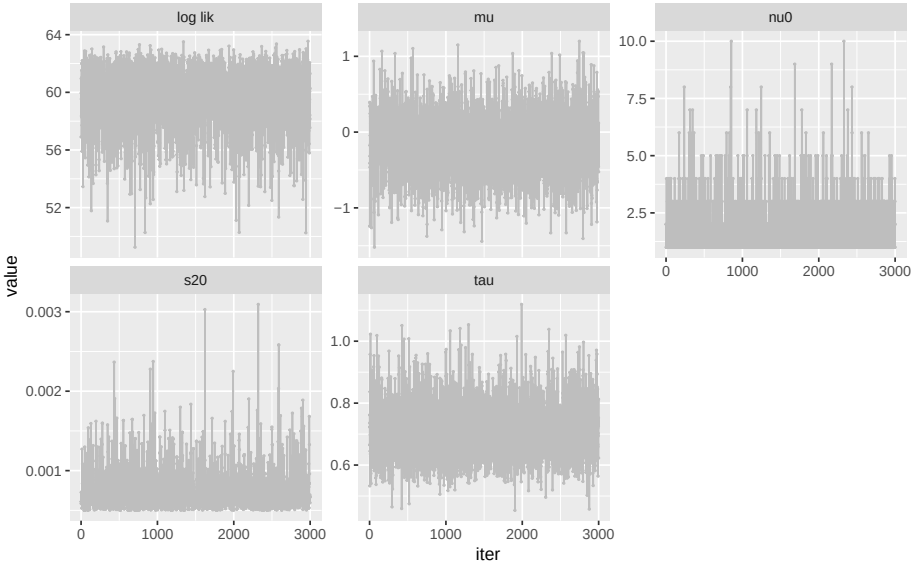
In the following codechunk, we visually inspect the chains for convergence.

```
model <- model.list[[1]]
model
## An object of class ' SingleBatchModel '
##   data:
```

```
##      n           : 35
##      k           : 2
##      table(z)    : 25, 10
##      mix prob (s): 0.7, 0.3
##      sigma (s)   : 0.01, 0.02
##      theta (s)   : -0.38, -0.01
##      sigma2.0 (s): 0
##      nu.0 (s)    : 1
##      logprior(s): -5.6
##      loglik (s)  : 84.44
chains <- ggChains(model)
chains[[1]]
```

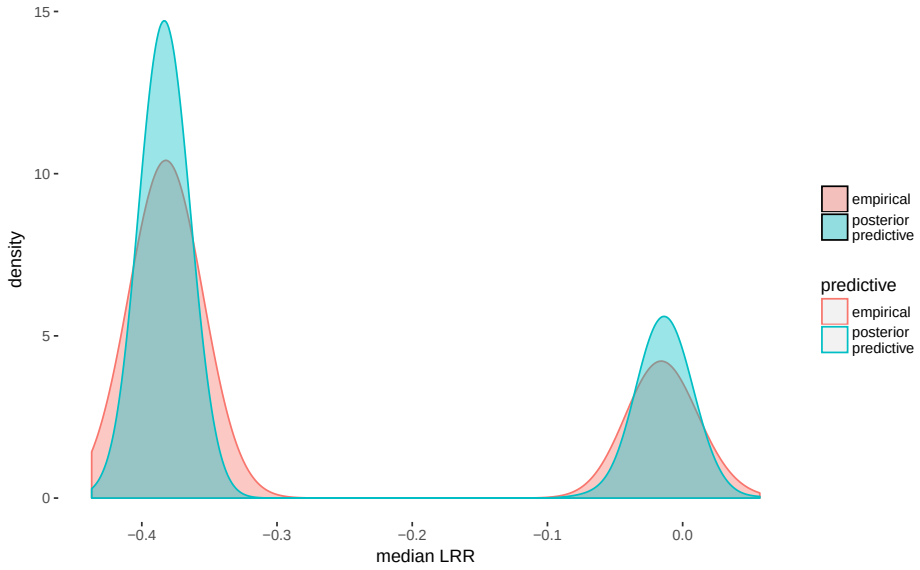



```
chains[[2]]
```



The posterior predictive distribution is also useful for assessing the adequacy of a model. Here, the approximation could be improved by increasing burnin, the number of iterations following burnin, and / or by increasing the thinning parameter.

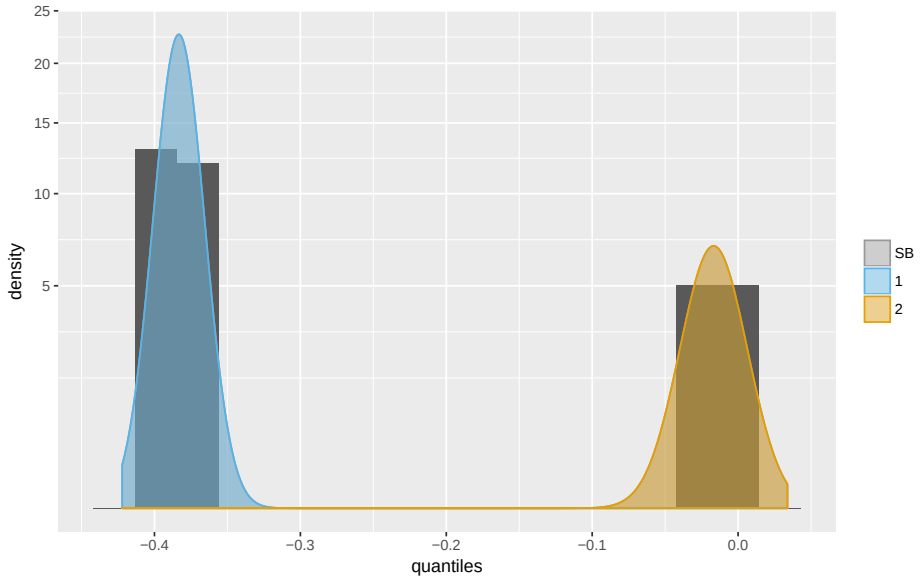
```
tab <- posteriorPredictive(model)
tab
## # A tibble: 1,000 x 2
##           y component
##       <dbl>     <int>
##  1 -0.386           1
##  2  0.0342          2
##  3  0.00767         2
##  4  0.0571          2
##  5 -0.373           1
##  6 -0.382           1
##  7 -0.396           1
##  8 -0.383           1
##  9 -0.357           1
## 10  0.0209          2
## # ... with 990 more rows
ggPredictive(model, tab) + xlab("median LRR")
```



2.4 Mapping mixture components to copy number states

While the MB and MBP implementations explicitly model batch effects, occasionally the log R ratios are skewed (non-Gaussian) for technical reasons other than batch effects. Unfortunately, there is no guarantee of a one-to-one mapping between mixture components and copy number. The function `CopyNumberModel` attemps to map copy number states to the mixture components by assessing the extent of overlap of the mixtures – the rationale being that two mixture components with substantial overlap are fitting the skewness of the data as opposed to multiple copy number states. Given additional information, one may also manually provide this mapping through the `mapping<-` function.

```
cn.model <- CopyNumberModel(model)
ggMixture(cn.model)
```



For additional details regarding model construction and mapping mixture components to copy number states, see Bayesian mixture models for copy number estimation. Having mapped copy number states to the mixture components, we can obtain the posterior probabilities for the copy number states:

```
probs <- probCopyNumber(cn.model)
head(probs)
##      [,1] [,2]
## [1,]    1    0
## [2,]    1    0
## [3,]    1    0
## [4,]    1    0
## [5,]    1    0
## [6,]    1    0
```

2.5 Big data

If thousands of samples are available, we generally do not need to fit the model to all samples in order to adequately estimate the mixture distribution. Below, we indicate a workflow for downsampling. First, we bin the data in the `MultiBatchModelExample`, requiring only 200 observations per batch. Data in small batches (less than 200 samples) are not binned. The `tileSummaries` function computes the average log R ratio for each bin.

```
mb <- MultiBatchModelExample
tiled.medians <- tileMedians(y(mb), 200, batch(mb))
tile.summaries <- tileSummaries(tiled.medians)
tile.summaries
## # A tibble: 718 x 3
```

```
##     tile avgLRR batch
##    <dbl>  <dbl> <int>
##  1     4  -1.35     1
##  2     6  -1.31     1
##  3     7  -1.30     1
##  4     9  -1.28     1
##  5    10  -1.27     1
##  6    11  -1.27     1
##  7    12  -1.26     1
##  8    13  -1.26     1
##  9    14  -1.25     1
## 10    16  -1.23     1
## # ... with 708 more rows
```

We fit the model to the mean summarized bins in the usual way. To speed up computation, we specify the true k=3 model in this toy example.

```
model <- gibbs(model="MB", k_range=c(3, 3),
               dat=tile.summaries$avgLRR,
               batches=tile.summaries$batch,
               mp=mp)[[1]]
## Fitting MB models
##   k: 3, burnin: 500, thin: 1
##      r: 1.01
##      eff size (minimum): 2844.8
##      eff size (median): 3038.2
##      marginal likelihood: -261.41
```

To map posterior probabilities back to the original observations, we use the function `upSample`.

```
model2 <- upSample(model, tiled.medians)
model2
## An object of class MultiBatchModel
##      n. obs      : 1500
##      n. batches  : 3
##      k           : 3
##      nobs/batch  : 500 500 500
##      loglik (s)  : -255.6
##      logprior (s): -7
```

The object `model2` can be converted to a copy number model using the same approach as described previously (i.e., `CopyNumberModel(model2)`). As there is a one-to-one mapping between mixture components and copy number, the posterior probabilities for the mixture components (`probz`) is the same as the posterior probabilties of the copy number states (`probCopyNumber`).

```
cn.model <- CopyNumberModel(model2)
mapping(cn.model)
## [1] 1 2 3
round(head(probz(cn.model)), 2)
##      [,1] [,2] [,3]
## [1,]    1    0    0
```

```
## [2,]    0    0    1
## [3,]    0    0    1
## [4,]    0    1    0
## [5,]    1    0    0
## [6,]    0    1    0
round(head(probCopyNumber(cn.model)), 2)
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    0    1
## [3,]    0    0    1
## [4,]    0    1    0
## [5,]    1    0    0
## [6,]    0    1    0
```

See the convergence vignette for assessing convergence and visually inspecting the MCMC chains for each parameter in the mixture model.

References

Barnes, Chris, Vincent Plagnol, Tomas Fitzgerald, Richard Redon, Jonathan Marchini, David Clayton, and Matthew E Hurles. 2008. "A Robust Statistical Method for Case-Control Association Testing with Copy Number Variation." *Nat Genet* 40 (10). Wellcome Trust Sanger Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge, UK.:1245–52. https://doi.org/10.1038/ng.206.

Cardin, Niall, Chris Holmes, Peter Donnelly, and Jonathan Marchini. 2011. "Bayesian Hierarchical Mixture Modeling to Assign Copy Number from a Targeted Cnv Array." *Genet. Epidemiol.* https://doi.org/10.1002/gepi.20604.

Eddelbuettel, Dirk, and Romain François. 2011. "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software* 40 (8):1–18. http://www.jstatsoft.org/v40/i08/.