Implementation of Bayesian mixture models for copy number estimation

Jacob Carey, Steven Cristiano, and Robert Scharpf

January 5, 2019

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). As described in the overview, four versions of the mixture model are implemented: SB, MB, SBP, and MBP. In each case, approximation of the posterior is by Markov Chain Monte Carlo (MCMC) written in C++ using the Rcpp package [@Rcpp].

For an EM-implementation of Gaussian mixture models for CNPs, see the Bioconductor package CNVtools [@Barnes2008]. 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] and implemented in the R package CNVCALL (http://niallcardin.com/CNVCALL).

The key distinction of this package is an emphasis on exploring models that account for batch effects – differences between groups of samples that were processed at different times and/or by different labs. For smaller studies, the more standard '`SingleBatchModel` may be reasonable. This package provides an infrastructure for fitting all of these models and then comparing these models by marginal likelihoods or Bayes factors. The main applications we envision these models for are in studies of germline diseases containing thousands of samples and several hundred CNPs. We suggest parallelizing by CNP (e.g., devoting a single core to each CNP), though leave the details on parallelization to the user.

```
library(CNPBayes)
library(ggplot2)
```

2 Implementation

CNPBayes uses several S4 classes to encapsulate key parameters of the MCMC simulation, reducing the need for functions with a large number of arguments and providing an explicit contract for the arguments passed to the C++ back-end. The three core classes are

- `McmcParams`: parameters for the number of burnin simulations (`burnin`), the number of independent chains to initialize (`nStarts`), the number of simulations after burnin (`iter`), and how often simulated values are to be saved (`thin`). For example, the following code indicates that we will run four independent chains (`nStarts = 4`) each with a burnin of 100 iterations. Following the burnin, we will save the first 1000 simulations and no simulations will be discarded for thinning.

```
mp <- McmcParams(iter=1000, burnin=100, thin=1, nStarts=4)
```

- `Hyperparameters`: a virtual class extended by `HyperparametersMultiBatch` and `HyperparametersSingleBatch` for the multi-batch (MB and MBP) and single-batch (SB and SBP) mixture model implementations. The MBP and SBP models have a pooled estimate of the variance across mixture components. Additional details are provided in the following section.

- `MixtureModel`: a virtual class with slots for data and hyperparameters, as well as a slot for each parameter. The class is extended by `SingleBatchModel` (SB), `MultiBatchModel` (MB), `SingleBatchPooled` (SBP) and `MultiBatchPooled` (MBP). S4 dispatch on these classes is used to handle MCMC updates that are specific to the single- and multi-batch models.
- After selecting a mixture model by some criteria (e.g., marginal likelihood or Bayes Factor), methods for inferring the copy number of the mixture components are handled by `SingleBatchCopyNumber` and `MultiBatchCopyNumber` classes.

3 Hyperparameters

Hyperparameters for the mixture model are specified as an instance of class `Hyperparameters`. Hyperparameters include:

- `k` the number of components. Defaults to 2.
- `mu.0` and `tau2.0` priors for $\mu \sim$ N(`mu.0`, `tau2.0`), the overall mean of components. Default to 0 and 0.4.
- `eta.0` and `m2.0` priors for $\tau^2 \sim$ Ga(shape=`eta.0`, rate=`m2.0`), the overall variance across components. (For the `MultiBatchModel` discussed in greater detail below, τ^2 is component-specific, representing the cross-batch variance of means for each component.) Defaults to 32 and 0.5.
- `alpha` the prior mixture probabilities. Does not have to sum to 1. Defaults to a k-length vector of 1's
- `beta` prior for ν_0. Defaults to 0.1.
- `a` and `b` priors for $\sigma_0^2 \sim$ Ga(shape=`a`, rate=`b`), the rate parameter for σ^2, the variance for each batch and component.

Instances of the hyperparameters class can be created by a constructor with the same name of the class. For example,

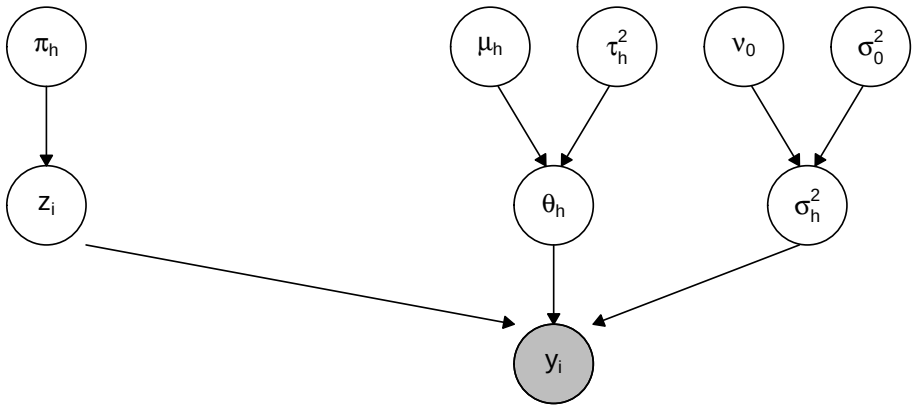
```
  Hyperparameters()
## An object of class 'Hyperparameters'
##    k      : 2
##    mu.0   : 0
##    tau2.0 : 0.4
##    eta.0  : 32
##    m2.0   : 0.5
##    alpha  : 1 1
##    beta   : 0.1
##    a      : 1.8
##    b      : 6
  HyperparametersMultiBatch()
## An object of class 'Hyperparameters'
##    k      : 3
##    mu.0   : 0
##    tau2.0 : 0.4
##    eta.0  : 32
##    m2.0   : 0.5
```

```
##    alpha  : 1 1 1
##    beta   : 0.1
##    a      : 1.8
##    b      : 6
```

`hpList` is a convenience function for creating a list of hyperparameter objects appropriate for each of the four types of models:

```
hp.list <- hpList()
names(hp.list)
## [1] "SB"  "MB"  "SBP" "MBP" "TBM"
hp.list[["SB"]] ## hyperparameters for SB model
## An object of class 'Hyperparameters'
##    k      : 2
##    mu.0   : 0
##    tau2.0 : 0.4
##    eta.0  : 32
##    m2.0   : 0.5
##    alpha  : 1 1
##    beta   : 0.1
##    a      : 1.8
##    b      : 6
hp.list <- hpList(k=3) ## each object will have hyperparameters for a 3-component mixture model
k(hp.list[["SB"]])
## [1] 3
```

A graphical overview of the model.



4 Simple example

CNPBayes allows for the simulation of test data. The number of observations, mixture proportions, means for each component, and standard deviations for each component must be specified.

```
set.seed(123)
sim.data <- simulateData(N=1000, p=rep(1/3, 3),
```

```
                              theta=c(-1, 0, 1),
                              sds=rep(0.1, 3))
```

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 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 – e.g, `k_range=c(1, 5)`). As described in the convergence vignette, an attempt is made by `gibbs` to adapt the thinning and burnin parameters to attain convergence. 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.

```
## Create McmcParams for batch model
mp <- McmcParams(iter=600, burnin=100, thin=1, nStarts=4)
model.list <- gibbs(model="SB", dat=y(sim.data),
                    k_range=c(3, 3), mp=mp)
## Fitting SB models
##   k: 3, burnin: 100, thin: 1
##      Gelman-Rubin: 21.27
##      eff size (median): 473
##      eff size (mean): 1480.3
##   k: 3, burnin: 200, thin: 3
##      Gelman-Rubin: 50.56
##      eff size (median): 786.8
##      eff size (mean): 1716
##   k: 3, burnin: 400, thin: 5
##      Gelman-Rubin: 30.06
##      eff size (median): 908.7
##      eff size (mean): 1935.2
##   k: 3, burnin: 800, thin: 7
##      Gelman-Rubin: 1.01
##      eff size (median): 1110.3
##      eff size (mean): 1215.2
##      marginal likelihood: -1018.32
```

We use Chib's estimator [@chib], running a reduced Gibbs sample to estimate the marginal likelihood of each model. Alternative methods for selection of k are also included. As an example, the `bic` method can be used for calculating the Bayesian Information Criterion. As the data is simulated from a three component distribution, `SB3` should be chosen.

```
round(sapply(model.list, marginal_lik), 1)
## SB3.SB3
## -1018.3
names(model.list)[1]
## [1] "SB3"
```

For comparing models of the same type, one could also use the BIC.

```
round(sapply(model.list, bic), 1)
##    SB3
```

```
## 2121.8
```

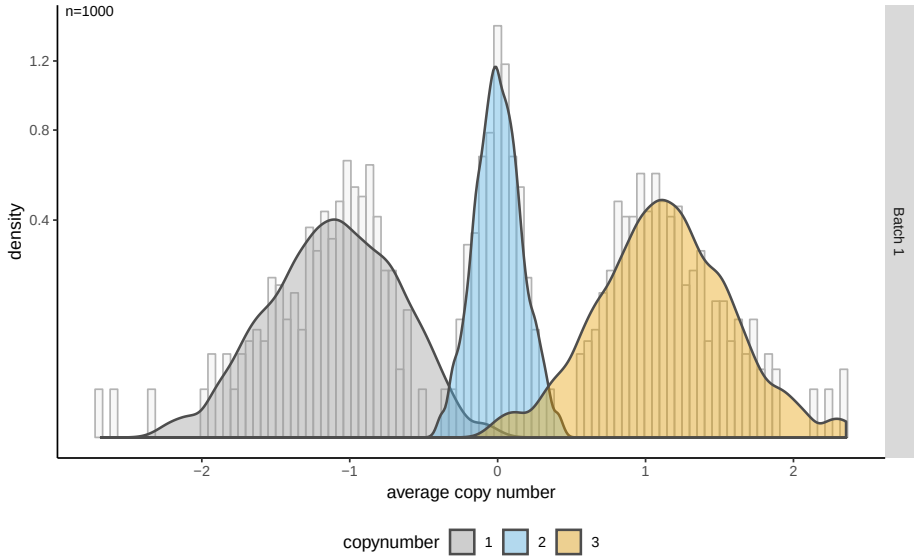
4.1 Copy number

The mixture components may not necessarily correspond to distinct copy number states. For example, if the number of probes in a CNP region is small, the average log R ratio calculated for each sample is more likely to be skewed. In practice, one would fit many models (e.g., k = 1 - 4), select the best model by the marginal likelihood, and then map components from the best model to distinct copy number states. Mapping mixture components to copy number states can be done manually using the `mapping<-` function. The constructor `CopyNumberModel` will automatically create a mapping based on the overlap of the component densities. By mapping mixture components to copy number states, the posterior probability can be computed for each copy number state as opposed to the mixture components.

```
model <- model.list[[1]]
cn.model <- CopyNumberModel(model)
mapping(cn.model)
## [1] "1" "2" "3"
```

The above mapping (1 2 3) proposes a one-to-one mapping between the 3 mixture components and the copy number states. Visualization of the model-based density estimates overlaying the data can be helpful for assessing the adequacy of this particular model:

```
ggMixture(cn.model)
```



As our interest is in the posterior probability of the copy number state (not the posterior probability of the mixture components), these probabilities can be computed from a copy number model by the function `probCopyNumber`.

```
head(probCopyNumber(cn.model))
##              [,1]      [,2]        [,3]
## [1,] 0.0000000000 0.0000000 0.999166667
```

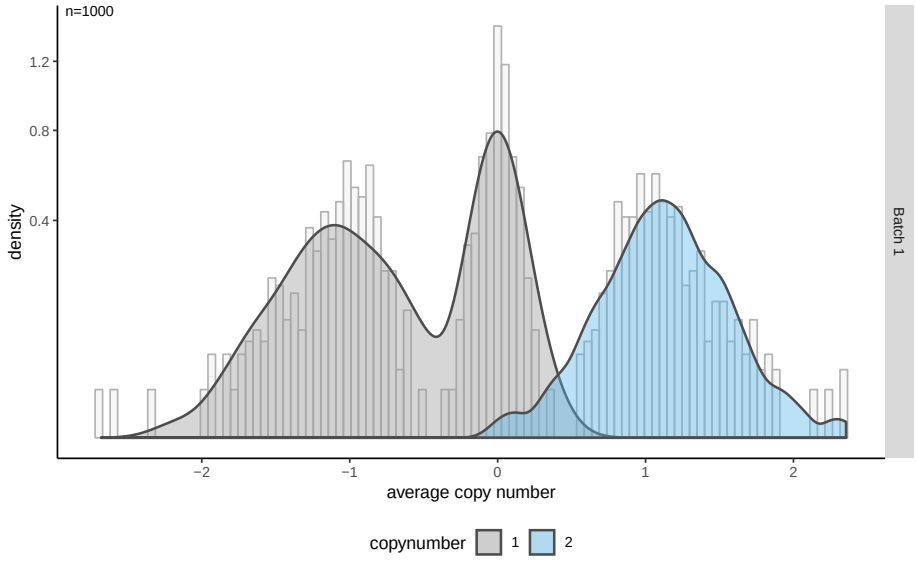
```
## [2,] 0.0000000000 0.0000000 0.999166667
## [3,] 0.0000000000 0.0000000 0.999166667
## [4,] 0.9991666667 0.0000000 0.000000000
## [5,] 0.0008326389 0.9941708 0.004163194
## [6,] 0.9991666667 0.0000000 0.000000000
```

One can explicitly set the mapping using the `mapping<-` replacement method. For example, here we map the first 2 components to the same state and the probability matrix now has only two columns:

```
cn.model2 <- cn.model
mapping(cn.model2) <- c("1", "1", "2")
head(probCopyNumber(cn.model2))
##           [,1]        [,2]
## [1,] 0.0000000 0.999166667
## [2,] 0.0000000 0.999166667
## [3,] 0.0000000 0.999166667
## [4,] 0.9991667 0.000000000
## [5,] 0.9950035 0.004163194
## [6,] 0.9991667 0.000000000
```

Again, `ggMixture` can be helpful for assessing the model. Here, mapping the first 2 components to the same state would only be reasonable if the differences between the first 2 components could be explained by batch effects. To evaluate this, one would want to identify the batches (next section) and fit the `MB` or `MBP` models in the `gibbs` function illustrated above.

```
ggMixture(cn.model2)
```



5 Batch Estimation

Differences in the one-dimensional summaries between groups of samples can arise due to technical sources of variation referred to as *batch effects*. For example, date, temperature, machine calibration, and lab technician are all potentially important technical sources of variation [@batch-effect]. Batch effects can vary by locus, with some loci more susceptible to technical factors that may affect measurement. The 96 well chemistry plate on which the samples were processed is often a useful surrogate for batch effects since this tends to capture samples that were processed (e.g., PCR amplification) and scanned at approximately the same time. However, in large studies involving potentially hundreds of chemistry plates, it is not computationally feasible to fit fully Bayesian plate-specific mixture models. Because chemistry plates are often processed at similar times and may be comparable in terms of the distribution of a statistic of interest, it may be more useful (and computationally scalable) to *batch* the chemistry plates. Here, we implement a simple two-sample Kolmogorov-Smirnov (KS) test for each pairwise combination of chemistry plates in the function `collapseBatch`. The two-sample KS test is applied recursively until no two combinations of grouped plates can be combined. We illustrate with a trivial example.

```
k <- 3
nbatch <- 3
means <- matrix(c(-1.2, -1.0, -1.0,
                  -0.2, 0, 0,
                  0.8, 1, 1), nbatch, k, byrow=FALSE)
sds <- matrix(0.1, nbatch, k)
N <- 500
sim.data <- simulateBatchData(N=N,
                              batch=rep(letters[1:3], length.out=N),
                              theta=means,
                              sds=sds,
                              p=c(1/5, 1/3, 1-1/3-1/5))
```

Again, one could apply the `gibbs` function to fit this data. Since we know the truth (same variance for each of the 3 mixture component and multiple batches), we could simply run the pooled variance multi-batch model (MBP) with 3 components:

```
gibbs(model="MBP", dat=y(sim.data), k_range=c(3, 3))
```

In practice, we would evaluate many model and select the best model by the marginal likelihood or the Bayes factor:

```
model.list <- gibbs(model=c("SBP", "MBP"), k_range=c(1, 5),
                    dat=y(sim.data),
                    batches=batch(sim.data), mp=mp)
```

6 References