BeviMed Guide

Daniel Greene

1 Introduction

BeviMed is a procedure for evaluating the evidence of association between allele configurations across rare variants within a genomic locus and a case/control label. It is capable of inferring the probability of association, and conditional on association, the probability of each mode of inheritance and probability of involvement of each variant. It works by applying Bayesian inference to two models indexed by γ. Under the model labelled $\gamma = 0$, the probability of the case label is independent of allele configuration at the given rare variant sites. Under the model labelled $\gamma = 1$, the probability of the case label is linked to the configuration of alleles, a mode of inheritance and a latent partition of variants into *pathogenic* and *non-pathogenic* groups.

The aim of the package is to facilitate prioritisation of large numbers of loci and variants therein by rapid inference of the posterior distributions of γ, mode of inheritance parameter m, and indicator of pathogenicity across variants, z. Unless otherwise stated, N refers to the number of individuals, k refers to the number of rare variants, m refers to the mode of inheritance (either m_{dom} or m_{rec}), and 'evidence' refers to the integrated likelihood of the data under a given model. The acronym 'MOI' will often be used to refer to mode of inheritance.

2 Functions and classes

`bevimed` is the key function in the package: it evaluates models $\gamma = 0$ and $\gamma = 1$ with respect to the input data, a logical length N vector of case/control labels y, and an $N \times k$ integer matrix of allele counts G. It evaluates model $\gamma = 0$ by computing the log evidence directly using the function `gamma0_evidence`. It performs inference on model $\gamma = 1$ conditional on each mode of inheritance separately by calling the function `bevimed_m` (where the '_m' is to be understood as 'conditional upon a given mode of inheritance'). The output of the function gives you the posterior distributions of interest:

- model indicator γ, i.e. $\mathbb{P}(\gamma = 1|y, G)$,
- mode of inheritance m given association, i.e. $\mathbb{P}(m|\gamma = 1, y, G)$,
- indicators of variant pathogenicity z_j for $j = 1, \ldots, k$ given mode of inheritance and association, i.e. $\mathbb{P}(z_j|m, \gamma = 1, y, G)$.

`bevimed` is simple to apply:

```
> obj <- bevimed(y=y, G=G)
```

It returns an object of class `BeviMed`, which contains the whole output of the inference. A summary of the inference can be printed by evaluating the object in an interactive session:

```
> obj
```

```
## ------------------------------------------------------------------------
## Posterior prob. of association: 0.038 [prior: 0.01]
## Posterior prob. of dominance given association: 0.798 [prior: 0.5]
## ------------------------------------------------------------------------
##                              dominant recessive
## Expected explained cases        4.557     1.107
## Expected explaining variants    2.438     2.437
```

```
## ------------------------------------------------------------------------
## Estimated probabilities of pathogenicity for individual variants
##
##  Var Ctrls Cases  Prob. dom pathogenic  Prob. rec pathogenic
##    1     0     3 [0.82 ===========   ] [0.57 =======       ]
##    2     0     1 [0.59 ========      ] [0.67 =========     ]
##    3     0     2 [0.69 =========     ] [0.59 ========      ]
##    4     1     0 [0.06               ] [0.56 =======       ]
##    5     0     1 [0.34 ====          ] [0.61 ========      ]
## ------------------------------------------------------------------------
```

It is a list containing slots:

- `"parameters"`, a list of parameter values used to call the function.
- `"moi"`, a list of `BeviMed_m` objects returned by the `bevimed_m` function, one for each mode of inheritance (i.e. dominant and recessive). The `BeviMed_m` class is a list containing samples from the posterior distributions of model parameters conditional on a given mode of inheritance (see help page `?bevimed_m` for more details). As a list, the MOI specific results can be looked up by MOI using the `$` operator, e.g. `x$moi$dominant`.

The function `bevimed_m` is an MCMC procedure which samples from the posterior distribution of parameters in model $\gamma = 1$. Each individual has an associated 'minimum number of alleles at pathogenic variant sites' required to have a *pathogenic configuration of alleles*. This is determined by the `min_ac` argument (defaulting to 1), and can be set to reflect the desired mode of inheritance. For example, in dominant inheritance, at least one pathogenic allele would render an allele configuration pathogenic, whilst for X-linked recessive inheritance, at least 1 and 2 pathogenic alleles would be required for a pathogenic configuration respectively for males and females. `bevimed` accepts a `ploidy` argument: an integer vector the same length as `y` which specifies the ploidy of each individual in the locus (defaulting to 2). Internally, it uses this argument to set `min_ac` automatically when it calls `bevimed_m` based on mode of inheritance.

Objects of class `BeviMed` typically take a large quantity of memory, so summarising with `summary` — which retains important summary statistics as a list — may be useful when performing multiple applications. Specific summary statistics can be obtained by looking them up in these summary lists (see help page `?summary.BeviMed` for names used for each statistic), or by calling an 'extract_' function on a `BeviMed`/`BeviMed_m` object:

- `extract_prob_association`: get the probability of association from a `BeviMed` object, optionally broken down by mode of inheritance by specifying `by_MOI=TRUE`,

```
> extract_prob_association(obj)

## [1] 0.0382511

> extract_prob_association(obj, by_MOI=TRUE)

##    dominant   recessive
## 0.030520093 0.007731005
```

- `extract_prob_pathogenic`: get marginal probabilities of pathogenicity for each variant from a `BeviMed` object,
- `extract_gamma1_evidence`: get the log evidence of model $\gamma = 1$ for a mode of inheritance from a `BeviMed_m` object,
- `extract_conditional_prob_pathogenic`: get a vector of probabilities of pathogenicity individual variants from a `BeviMed_m` object.

```
> extract_conditional_prob_pathogenic(obj$moi$dominant)

## [1] 0.82000000 0.58777778 0.69444444 0.05888889 0.33555556
```

- `extract_expected_explained`: get the expected number of cases with a pathogenic configuration of alleles from a `BeviMed_m` object,
- `extract_explaining_variants`: get the expected number of pathogenic variants for which cases harbour rare alleles from a `BeviMed_m` object.

Each of these functions has an equivalent one by the same name without the 'extract_' prefix which can be called with the same raw arguments as `bevimed`/`bevimed_m`: $y, G, \ldots$, etc.

```
> prob_association(y=y, G=G)
```

```
## [1] 0.03760044
```

Note that the result of calling `prob_association` is slightly different due to Monte Carlo error as the inference procedure has been repeated.

`bevimed` passes arguments to `bevimed_m` through the '`...`' argument. However, sometimes it is preferable to pass different arguments to `bevimed_m` depending on mode of inheritance. `bevimed` therefore allows mode of inheritance specific arguments to be passed through `dominant_args` and `recessive_args`, which should be named lists of arguments then only used in the corresponding calls to `bevimed_m`. For example, it might be thought that fewer variants would be linked to disease given a dominant mode of inheritance than would given recessive inheritance, in which case `dominant_args` could be used to pass a prior with a lower mean to the dominant application of `bevimed_m`.

3 Priors on model parameters

The user can control the prior distributions of the model parameters when applying the inference functions `bevimed`, `bevimed_m` and `gamma0_evidence` as listed below.

- The probability of association, $\mathbb{P}(\gamma = 1|y)$, with argument `prior_prob_association` in the `bevimed` function (defaults to 0.01).
- The probability of dominant inheritance given association, $\mathbb{P}(m = m_{\text{dom}})$, with the `prior_prob_dominant` in the `bevimed` function (defaults to 0.5).
- The hyper parameters of the Beta prior for the probability τ_0 of observing the case label under model $\gamma = 0$. Values for the hyper parameters can be passed to the `bevimed` and `gamma0_evidence` functions as the `tau0_shape` argument (defaults to a vague parameterisation of $\alpha = \beta = 1$).
- The hyper parameters of the Beta prior for τ and π, respectively the probabilities of observing the case label for individuals with non-pathogenic and pathogenic allele configurations under model $\gamma = 1$. The default for τ is the same as for τ_0, but the default for π has a mean close to 1, as typically for rare diseases the variants are high penetrance, i.e. have a high probability of causing the disease phenotype. Values for these hyper parameters can be passed as arguments `tau_shape` and `pi_shape` to the `bevimed` and `bevimed_m` functions.
- The prior on the indicators of variant pathogenicity, z. By default, all variants have a shared prior on their probability of pathogenicity, $z_j \sim \text{Bernoulli}(\omega)$ with $\omega \sim \text{Beta}(\alpha = 2, \beta = 8)$. The hyper parameters for ω can be specified by the user using the parameter `omega_shape`. However the user can also control the prior on pathogenicity for individual variants. This is done using the `variant_weights` parameter, a numeric vector of length k labelled c in the model specification. The effect of the c values is given by the logistic equation:

$$\begin{aligned} z_j &\sim \text{Bernoulli}(p_j), \\ \operatorname{logit} p_j &= \omega + \phi c_j, \\ \log \phi &\sim \text{N}(\mu_\phi, \sigma_\phi^2), \end{aligned}$$

  where ϕ is the scaling factor for c. By default, c is centralised on 0 so that ω is interpretable as the global rate of pathogenicity in the locus, and ϕ has a mean of 1, so c_j is interpretable as a shift in the log odds on the prior probability of variant j being pathogenic. The raw values of c as given in the `variant_weights` arguments will be used if the parameter `standardise_weights` is set to `FALSE`. The hyper parameters μ_ϕ and

σ_ϕ for the prior distribution of $\log \phi$ are respectively represented by arguments `log_phi_mean` and `log_phi_sd`. Hyper parameters for ω and ϕ and the values for c can be passed to functions `bevimed` and `bevimed_m`.

Estimating the scaling factor ϕ in this way has the advantage of maintaining power even when the weights are counter-productive, as ϕ can take values close to 0 making the weights redundant. However, it is possible to make the effect of variant weights c fixed by setting the parameter `estimate_phi` to `FALSE`, in which case ϕ is fixed at 1.

4 Application to real data

It is an assumption of model $\gamma = 1$ that alleles are *identitical by state* rather than by descent. This would typically be the case if only unrelated individuals are included in the analysis, and variants are filtered for low allele frequency across all ethnic groups. It is therefore recommend to take these steps in order to set G. Various software is available for performing these tasks: as an example 'SAMtools' and 'KING' can be used for variant filtering and inferring relatedness respectively. There is also various software for reading VCF files into `R`. The 'BeviMed with VCFs' vignette contains instructions on how to read allele counts across variants in a given locus into `R` from a VCF file directly as a matrix using simple functions depending on the program 'tabix'. However, although this method could be effective for testing a single locus, typically testing association between a disease and multiple loci is required, in which case reading variants belonging to multiple loci at the same time is likely to be more efficient. Often, it will be most effective to read data for as many variants as possible into memory (e.g. breaking up the VCF by chromosome), and looping through loci one at a time, applying `bevimed` the allele count matrix of its variants. Typically loci would correspond to genes, but it is also applicable to non-coding loci, for example, transcription factor binding sites. In order to increase power, variants which are unlikely to be involved in disease can be filtered out, or have their probability of pathogenicity down-weighted using the `variant_weights` parameter. For example, synonymous variants could be removed, and loss-of-function variants could be up-weighted. One could also create multiple sets of variants corresponding to a single locus: for example, a set containing only loss-of-function variants and a set containing all loss-of-function, missense and UTR variants. One could then assign a prior probability of association to each set, and the posterior probability of association with each set could then be inferred by computing the evidence for each one in turn and combining with the prior probabilities.

Although typically testing association between a disease and multiple loci is required, `BeviMed` only provides procedures for dealing with a single locus. This is because most of the time such an analysis is computationally expensive due to the large number of applications required or large quantity of genetic data which must be loaded, and full control is required in order to best exploit the resources available. Here we provide a simple example script which applies the inference to multiple loci and tabulates the results with columns for gene name, posterior probability of association and probability of dominant inheritance given the association. Let `chr1genes` be a `data.frame` of chromosome 1 genes with columns for name, start position and end positon (the 'biomaRt' package could be used to obtain such a table), and `y` be a logical vector indicating disease status, the same length as the number of samples in the VCF.

```
> source(paste0(system.file(package="BeviMed", "/scripts/vcf.R")))
> all_variants <- vcf2matrix("my-vcf.vcf.gz", chr="1", from=1, to=1e9, include_variant_info=TRUE)
> row_indices_per_gene <- lapply(1:nrow(chr1genes), function(i) {
+         which(all_variants$info$POS >= chr1genes$start[i] & all_variants$info$POS <= chr1genes$end[i])
+ })
> names(row_indices_per_gene) <- chr1genes$gene
>
> results <- mclapply(
+         mc.cores=16L,
+         X=chr1genes$gene,
+         FUN=function(gene) {
+                 G <- all_variants$G[variant_inds[[gene]],,drop=FALSE]
+                 c(
+                         list(gene=gene),
+                         summary(bevimed(y=y, G=G))) })
>
> results_table <- do.call(what=rbind, lapply(results, function(x) data.frame(
```

```
+         Gene=x[["gene"]],
+         `Prob. assoc`=sum(x[["prob_association"]]),
+         `Prob. dominance`=x[["prob_association"]]["dominant"]/sum(x[["prob_association"]]),
+         check.names=FALSE,
+         stringsAsFactors=FALSE
+ )))
```

5 Performance and tuning

As an MCMC based procedure, statistics produced by `bevimed` have Monte Carlo error. In the implementation in the `BeviMed` package, z is the only parameter which is sampled and is updated using Gibbs sampling of each component z_j in turn. If variant weights are included, ω and ϕ are also sampled using Metropolis-Hastings within Gibbs steps, causing estimates of the evidence to have higher variance for the same number of samples. By default, `bevimed` draws $1,000$ samples from each of 7 tempered chains in the MCMC algorithm, running at temperatures $t = \left(\frac{l}{6}\right)^2$ for $l \in \{0, 1, \ldots, 6\}$. We have found that this parameterisation leads to quick execution and stable results for sample sizes up to $5,000$ and loci containing over $2,000$ variants, also allowing for the inclusion of variant weights. However, if much larger sample sizes or larger numbers of variants are used — particularly if variant weights are included — it may become necessary to modify the parameters controlling the sampling routine in order to improve the accuracy of the results. Strategies for doing this include:

- increase the number of samples drawn per tempered chain using the `samples_per_chain` argument,
- increase the number tempered chains or change the distribution of temperatures using the `temperatures` argument,
- pass the `tune_temps` argument to `bevimed`, specifying the number of temperatures to select by interval bisection for use in the final application,
- if estimating ϕ and ω, set `tune_omega_and_phi_proposal_sd=TRUE` in the call to `bevimed` in order to adaptively tune the standard deviations of the Metropolis-Hastings proposal distributions so that the acceptance rate falls within a given range, defaulting to $[0.3, 0.7]$. If this option is used, a tuning run of the MCMC algorithm is applied, which estimates a proposal standard deviation for each temperature using successive blocks of `tune_block_size` samples until the desired acceptance rate is obtained.

It is also possible to instruct `bevimed_m` to halt sampling once the estimated evidence lies within a given confidence interval, or once there is sufficient confidence that the evidence is greater than some threshold. The latter might be useful, for instance, if many regions were being tested for association and only those with very strong evidence for association were of interest). By default, `bevimed_m` does not attempt to stop sampling, and always draws `samples_per_chain` samples for each tempered chain. In terms of the argument names, by setting `stop_early=TRUE`, `bevimed_m` draws up to `blocks` batches of `samples_per_chain` samples, stopping as soon as the estimated log evidence lies within a confidence interval of width `tolerance` (defaults to 1) with confidence of `confidence` (defaults to 0.95) based on `simulations` simulations (defaults to $1,000$), or as soon as there is `confidence` confidence that it is below `log_evidence_threshold`.