BeviMed Guide

Daniel Greene

1 Introduction

BeviMed [1] is a procedure for evaluating the evidence of association between allele configurations across rare variants, typically within a genomic locus, and a case/control label. It is capable of inferring the posterior 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 model comparison between to two models indexed by γ. Under the model labelled $\gamma = 0$, the probability of case status is independent of allele configuration at the given rare variant sites. Under the model labelled $\gamma = 1$, the probability of case status is linked to the configuration of alleles, and a latent partition of variants into *pathogenic* and *non-pathogenic* groups conditional on a mode of inheritance. It can also compare several alternative models, each of which includes a different subset of variants. This has the effect of inducing a prior correlation on the variant pathogenicities, which can boost power if only a particular (though unknown) class of variants is responsible for disease.

The aim of the package is to facilitate prioritisation of large numbers variant sets, typically drawn from different loci, by rapid inference of the posterior distributions of γ, mode of inheritance parameter m, and indicator of pathogenicity across variants, z. This guide describes the interface of the package in detail, relating it to the underlying statistical model and including some implementation details. See the 'BeviMed Introduction' vignette for a quick start guide. The description is given in terms of the paper, Greene et al. (2017) [1], although it is not necessary to read the paper in order to follow it. 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` has functions for evaluating models $\gamma = 0$, $\gamma = 1$ with $m = m_{\text{dom}}$, and $\gamma = 1$ with $m = m_{\text{rec}}$ with respect to the data — a logical length N vector of case/control labels y, and an $N \times k$ integer matrix of allele counts G:

- `gamma0_evidence`, which computes the evidence for model $\gamma = 0$, $\mathbb{P}(y|\gamma = 0)$.
- `bevimed_m`, which samples from the posterior distribution of the model $\gamma = 1$ conditional on a given mode of inheritance. The output yields the evidence $\mathbb{P}(y|m, \gamma = 1, G)$ and probabilities of variant pathogenicity, $\mathbb{P}(z_j|m, \gamma = 1, y, G)$ for $j = 1, \ldots, k$.
- `bevimed`, which evaluates all three models in turn by calling `gamma0_evidence` and `bevimed_m` with respect to each mode of inheritance. By allowing prior probabilities and computing the evidence for each model, it allows the posterior quantities of interest to be computed using Bayes' theorem.
  - model indicator γ, $\mathbb{P}(\gamma = 1|y, G)$
  - mode of inheritance m given association, $\mathbb{P}(m|\gamma = 1, y, G)$.
- `bevimed_polytomous`, which evaluates model $\gamma = 0$ and an arbitrary number of association models using `bevimed_m`, corresponding to different subsets of variants within G and given modes of inheritance.

`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 probability of association:
##  0.038 [prior: 0.01]
## ------------------------------------------------------------------------------------
##      Model MOI Prior  Post Cases Variants
##   dominant dom   0.5 0.799  4.56     2.44
##  recessive rec   0.5 0.201  1.13     1.61
##
## MOI: mode of inheritance, dominant (dom) or recessive (rec)
## Prior: prior probability of model given association
## Post: posterior probability of model given association
## Cases: posterior expected number of cases explained
## Variants: posterior expected number of variants involved in explained cases
## ------------------------------------------------------------------------------------
## Probabilities of pathogenicity for individual variants given association
##
##  Var   Probability pathogenic
##    1 [0.77 =============    ]
##    2 [0.60 ==========       ]
##    3 [0.68 ============     ]
##    4 [0.16 ===              ]
##    5 [0.39 =======          ]
## ------------------------------------------------------------------------------------
```

An object of class `BeviMed` is a list containing slots:

- `"parameters"`, a list of parameter values used to call the function.
- `"models"`, a list of `BeviMed_m` objects returned by the `bevimed_m` function, one for each association model - typically one per 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 model specific results can be looked up by model using the `$` operator, e.g. `x$models$dominant`.

The function `bevimed_m` uses an MCMC algorithm to sample from the posterior distribution of the 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, and thus the `min_ac` argument could be set to 1. In X-linked recessive inheritance, at least one and two pathogenic alleles would be required for a pathogenic configuration respectively for males and females, and thus the `min_ac` argument could be given as a numeric vector of length N with 1s for males and 2s for 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.

When phased genotype data is available, separate genotype matrices for each haplotype maybe supplied (i.e. as well as specifying `G`, passing an additional matrix `G2` - which defaults to `NULL`) to the relevant function. If `G2` is not null, BeviMed is run in phased mode, in which case, when run under a recessive model (i.e. `min_ac = ploidy`), at least one pathogenic allele on each haplotype are required to give an individual a pathogenic configuration.

Objects of class `BeviMed` typically consume a large amount of memory. Summarising objects of class `BeviMed` 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).

`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 for the parameter to the dominant application of `bevimed_m`.

Similarly, `bevimed_polytomous` evaluates association models depending on variants corresponding to those in the given `variant_sets` argument: a list of `integer` vectors, one for each model, each indexing variants with respect to their column position in G. Arguments are passed to `bevimed_m` through the '`...`' argument, as with `bevimed`, but model specific arguments are passed using the parameter `model_specific_args`: a list of the same length as `variant_sets`. The mode of inheritance parameter for each association model can be specified as using a character vector the same length as `variant_sets` containing elements "dominant" or "recessive" (defaults to `"dominant"`). The prior probability of association for each model can also be specified as a numeric vector of probabilities using the argument `prior_prob_association`.

In this example we use `bevimed_polytomous` to compare two models in the situation where the disease status depends on only the allele configuration at the first variant site from amongst 5 columns: one depending on just the first variant and one depending on all the variants.

```
> bevimed_polytomous(y=G[,1] > 0, G=G, variant_sets=list(`first`=1, `all`=1:ncol(G)))

## ------------------------------------------------------------------------------
## Posterior probability of association:
##  0.132 [prior: 0.01]
## ------------------------------------------------------------------------------
##  Model MOI Prior  Post Cases Variants
##  first dom   0.5 0.721  2.91    0.969
##    all dom   0.5 0.279  2.82    0.941
##
## MOI: mode of inheritance, dominant (dom) or recessive (rec)
## Prior: prior probability of model given association
## Post: posterior probability of model given association
## Cases: posterior expected number of cases explained
## Variants: posterior expected number of variants involved in explained cases
## ------------------------------------------------------------------------------
## Probabilities of pathogenicity for individual variants given association
##
##  Var   Probability pathogenic
##    1 [0.96 ================ ]
##    2 [0.01                  ]
##    3 [0.00                  ]
##    4 [0.01                  ]
##    5 [0.01                  ]
## ------------------------------------------------------------------------------
```

3 Priors on model parameters

The user can control the prior distributions of the model parameters when applying the inference functions `bevimed`, `bevimed_polytomous`, `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:

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

  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. Thus, one could for example of use the untransformed CADD Phred score [2] for each variant as a weight. 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 variants are not linked across loci. We therefore recommend removal of any first, second and third degree relatives, and filtering variants are for low allele frequency across all ethnic groups before applying the function. 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. It is also straightforward to apply the inference to multiple sets of variants corresponding to different classes of variants for a single locus using the `bevimed_polytomous` function. For example, a set containing only loss-of-function variants could be used to evaluate evidence for association between the disease and a 'knocked-out' gene. This increases power if the only variants of a particular class increase disease risk. Prior probabilities of association with each model set can then be combined with the evidence to obtain the posterior probability of association with each model/variant set.

Although typically testing association between a disease and multiple sets of variants is required, `BeviMed` only provides procedures for dealing with a single set of variants at a time. This is because such analyses are often

computationally expensive due to targeting a large number of sets of variants or involve a large volume of genetic data, and full control of the distribution of jobs between CPUs and compute nodes 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. Note that it may be necessary to read the VCF data into R in smaller parts, e.g. by modifying the `from` and `to` arguments of `vcf2matrix` accordingly, so as not to overwhelm the memory.

```
> 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[row_indices_per_gene[[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 from `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`.

By default the function `bevimed_m` stores the complete set of samples drawn during the MCMC process (after burn-in samples are removed) and therefore this function typically uses lots of memory. The vast majority of memory usage is expended storing the trace of samples of z, the indicator of pathogenicity for each variant) and x, the indicator of having a pathogenic configuration for each individual. Storing these traces enables useful summary statistics — for example the expected number of explained cases — to be computed from the output. However, users may only be interested in the probability of association, at least in the first application. In this case, the arguments `return_x_trace` and `return_z_trace` can be set to `FALSE` when calling `bevimed_m` in order to conserve memory, but still allow the evidence and probabilities of association to be computed.

The inference functions `bevimed_m`, `bevimed` and `bevimed_polytomous` always sample from the posterior distribution of pathogenicity of each variant represented in allele count matrix G. However, G does not necessarily contain data relevant to pathogenicity for all variants which are represented in it. This occurs when the allele counts for a variant are zero for all individuals, or when conditioning on recessive inheritance and alleles for the variant are only present for individuals whose total allele count is less than their ploidy. The function `subset_variants` can be used to remove such variants, returning either a transformed matrix or the indices of the variants in the original set for which there is data relevant to pathogenicity in G. Typically, G would only contain variants which were observed in at least one of the individuals, so using this function *a priori* is not likely to result in a speed up when applied conditioning on dominant inheritance, as no variants would be removed. However, it is often the case that only a small number of variants are observed in compound heterozygotes/homozygotes, so it is likely to result in a speed up conditioning on recessive inheritance.

References

[1] Greene D, NIHR BioResource, Richardson S, Turro E. (2017). A Fast Association Test for Identifying Pathogenic Variants Involved in Rare Diseases. American Journal of Human Genetics, 101(1):104–114. doi: 10.1016/j.ajhg.2017.05.015.

[2] Kircher M, Witten DM, Jain P, O'Roak BJ, Cooper GM, Shendure J. (2014). A general framework for estimating the relative pathogenicity of human genetic variants. Nature Genetics, 46(3):310–315. doi: 10.1038/ng.2892.