HMMbvs-vignette

Mingrui Liang, Matthew D. Koslovsky, Emily T. Hebert, Darla E. Kendzor, Michael S. Businelle, Marina Vannucci

In this tutorial, we demonstrate how to implement Bayesian variable selection for both HMMs and MSMs using our contributed R package `HMMbvs`. Since the true relations between covariates and outcomes are unknown in practice, we chose to apply our model to simulated data in order to display its selection performance.

Install the package

Prior to installing the `HMMbvs` package, make sure the following packages are installed and loaded in R: `Rcpp`, `RcppArmadillo`, `msm`, `MASS`, `coda`, `reshape`, `abind`, `devtools`, `ggplot2`. Then to install `HMMbvs`, run

```
devtools::install_github("mliang4/HMMbvs")
```

in the R console. Once the package is installed, it is loaded into the R environment by simply running

```
library(HMMbvs)
```

Data

In this section, we describe the data structure required to run our proposed method using the `HMMbvs_R()` function. Data are inputted into the function as a `data.frame` object in the long format, where each row corresponds to the j^{th}, $j = 1, \dots, n_i$, observation from the i^{th}, $i = 1, \dots, \dot{N}$, subject. Note that the `HMMbvs_R()` function requires the observations for each subject to be in chronological order, but the order of the subjects does not matter. To run, the dataset must contain (at the very least) three columns representing the subject ID, `"id"`, observed state, `"y"`, and observation time, `"obstime"` for each observation. While there is no specific order for the columns in the dataset, the subject IDs should be numeric, the observed state should only take on values of 0 or 1, and the observation times should be positive real numbers. Additional columns in the dataset represent the corresponding covariates collected at the ij^{th} assessment. Note that these can be both baseline and time-varying covariates. Prior to analysis, categorical covariates should be transformed into indicator variables and labeled appropriately.

Our package includes functionality to simulate data similar to our simulation and sensitivity analysis, which we use to demonstrate how to apply our method. To simulate data, run

```
set.seed(125)
df = sim_data( ind = 150, window = 30, lambda_0 = 0.5, mu_0 = 0.5, emisP0 = 0.95, emisP1 = 0.95,
      pt = 30, pe = 20, bval = c(0.7,1.0,1.5), biased = TRUE )
```

Here, `set.seed(125)` seeds the random number generator to make the simulation reproducible. By default, a dataset with 150 individuals (`ind = 150`), each with 30 transitions (`window = 30`), is generated with 30 transitions and 20 emissions covariates (`pt = 30, pe = 20` respectively). The first four transitions covariates and the first two emissions covariates are binary. The rest of the covariates are simulated from a normal distribution and standardized. This implies a total of 100 potential regression terms across both levels of the model, not including intercept terms. In the true model, 20% of the regression terms are active, and their effects are randomly chosen from ± 0.7, ± 1.0 or ± 1.5 (`bval = c(0.7,1.0,1.5)`). The baseline hazard rates for transitions in both directions are set to 0.5 by default (`lambda_0 = 0.5, mu_0 = 0.5`). The baseline probability of reporting a true non-smoking or smoking behavior (β_0 and β_1, respectively) are set to $logit(.95)$, reflecting a 95% accuracy level for both non-smoking and smoking states (`emisP0 = 0.95, emisP1 = 0.95`). Naturally, our simulated data will be biased (`biased = TRUE`) by default. We can set this argument to (`biased = FALSE`), to replace the biased observation with the true hidden state sequence (i.e. 100% accurate observation). The following 11 rows of output from the simulated `df` object display the last six observations for subject ID 1 and the first five observations for subject ID 2 simulated by the `sim_data()` function

```
print(df$DATA[26:36,c(1,2,3,31,32,33,51,52,53)],row.names = F)
#>  id tVar1 tVar2     tVar30 eVar1 eVar2     eVar20 y    obstime
#>   1     0     0 -0.9601369     0     0  1.2491446 1 27.7213073
#>   1     0     0 -0.9601369     0     0  1.2491446 1 28.2577059
#>   1     0     0 -0.9601369     0     0  1.2491446 1 28.5202212
#>   1     0     0 -0.9601369     0     0  1.2491446 1 28.5304124
#>   1     0     0 -0.9601369     0     0  1.2491446 0 29.8044472
#>   1     0     0 -0.9601369     0     0  1.2491446 1 29.8986442
#>   2     0     0 -1.3292270     0     0 -0.4940494 1  0.0000000
#>   2     0     0 -1.3292270     0     0 -0.4940494 0  0.2207321
#>   2     0     0 -1.3292270     0     0 -0.4940494 0  1.7897745
#>   2     0     0 -1.3292270     0     0 -0.4940494 0  2.0517698
#>   2     0     0 -1.3292270     0     0 -0.4940494 0  2.3852817
```

where `df$DATA` extracts the simulated data from the `df` object. Note that the simulated covariates for transitions `"tVar3"` to `"tVar29"` and emissions `"eVar3"` to `"eVar19"` are suppressed for demonstration purposes. The output of `sim_data()` also contains the true regression coefficients and the true hidden state sequence. They can be extracted by running

```
trueBeta = df$BETA
trueState = df$STATE
```

An unbiased dataset can be generated by simply changing the `biased` argument to `FALSE` as follows:

```
set.seed(125)
df1 = sim_data( ind = 150, window = 30, lambda_0 = 0.5, mu_0 = 0.5, emisP0 = 0.95, emisP1 = 0.95,
      pt = 30, pe = 20, bval = c(0.7,1.0,1.5), biased = FALSE )
```

```
print(df1$DATA[26:36,c(1,2,3,31,32,33,51,52,53)],row.names = F)
#>  id tVar1 tVar2     tVar30 eVar1 eVar2     eVar20 y    obstime
#>   1     0     0 -0.9601369     0     0  1.2491446 1 27.7213073
#>   1     0     0 -0.9601369     0     0  1.2491446 1 28.2577059
#>   1     0     0 -0.9601369     0     0  1.2491446 1 28.5202212
#>   1     0     0 -0.9601369     0     0  1.2491446 1 28.5304124
#>   1     0     0 -0.9601369     0     0  1.2491446 1 29.8044472
#>   1     0     0 -0.9601369     0     0  1.2491446 1 29.8986442
#>   2     0     0 -1.3292270     0     0 -0.4940494 1  0.0000000
#>   2     0     0 -1.3292270     0     0 -0.4940494 0  0.2207321
#>   2     0     0 -1.3292270     0     0 -0.4940494 0  1.7897745
#>   2     0     0 -1.3292270     0     0 -0.4940494 0  2.0517698
#>   2     0     0 -1.3292270     0     0 -0.4940494 0  2.3852817
```

We can see that the observation y, which is the true state sequence here, is different from the one in the previous dataset.

Implementation of the MCMC Algorithm

To generate MCMC samples for the HMM applied to the data simulated in the previous section, run

```
hmmout = HMMbvs_R( data = df$DATA,
           tcova = c( "tVar1", "tVar2", "tVar3", "tVar4", "tVar5", "tVar6", "tVar7", "tVar8",
                      "tVar9", "tVar10", "tVar11", "tVar12", "tVar13", "tVar14", "tVar15" ),
           ecova = c( "eVar1", "eVar2", "eVar3", "eVar4", "eVar5", "eVar6", "eVar7" ),
           tforce = NULL, eforce = NULL, standardize = NULL, model = "HMM", init = "baseline",
           initvalue = NULL, iter = 25000, v1 = 5, v2 = 1, a = 1, b = 9, thin = 10, thin_hidden = 10
           base01larger=-1, base00larger=-1)
```

The first three arguments are required for the model to run. In this example, `data = df$DATA` indicates the dataset to be analyzed. The `tcova` and `ecova` arguments correspond to the column names in the dataset to include in the transitions and emissions parts of the the model, respectively. They are set to a list of zero length (the `NULL` object in `R`) by default and they accept either a vector or a named list of two vectors as input. This allows the user to control which covariates may be associated with different parts of the model. For example, even though there are 30 transitions and 20 emissions covariates included in the `df$DATA` dataset, we only include `tVar1` to `tVar15` and `eVar1` to `eVar7` in the model. If a vector is supplied, as in our example, then the covariates named in `tcova` may affect both transitions (i.e., $0 \rightarrow 1$ and $1 \rightarrow 0$). Similarly, all of the covariates in `ecova` are included in both emissions models. If it is of interest to include different covariates for each of the transitions or emissions components in the model, then a list of two vectors, can be provided. For example, the following setup

```
tcova = list( "t01" = c( "tVar1", "tVar2", "tVar3", "tVar4", "tVar5", "tVar6" ),
              "t10" = c( "tVar3", "tVar4", "tVar5", "tVar6" ) )
ecova = list( "e00" = c( "eVar1", "eVar2", "eVar3", "eVar4" ),
              "e11" = c( "eVar2", "eVar4", "eVar6" ) )
```

implies that only covariates `tVar` 1-6 may influence transitions from state 0 to 1, `"t01"`, and covariates `tVar` 3-6 may influece transitions from state 1 to 0, `"t10"`. Similarly, `e00` and `e11` indicate the covariates potentially associated with accurately reporting outcomes. By default, all items provided in `tcova` and `ecova` are subject to selection. Users can force covariates into the model by using the `tforce` and `eforce` arguments. For example,

```
tforce = list( "tf01" = c( "tVar2", "tVar4" ),
               "tf10" = c( "tVar4", "tVar7", "tVar8" ) )
eforce = list( "ef00" = c( "eVar2" ) )
```

forces `tVar` 2 and 4 to be associated with transitions from state 0 to state 1 and `tVar` 4, 7, and 8 to be associated with transitions from 1 to 0. These arguments can handle a named list of size one or two as input. For example, `eVar2` is forced into the model for emissions from state 0, but all covariates are free to be selected in the model for emissions from state 1. In practice, we suggest that all continuous covariates are standardized prior to analysis. Users can provide a vector of variable names to standardize via the `standardize` argument.

Our function can run both HMM (`model = "HMM"`) or MSM (`model = "MSM"`), with the HMM run by default. Additionally, the user has control over how the MCMC sampler is initialized. By default, non-intercept regression coefficients will be initiated at 0 by `init = "baseline"`. Alternatively, one can use `init = "warmstart"` to initiate the regression terms at the maximum likelihood estimates provided by the `msm` package in `R`. The model can also be initialized with manual input (`init = "manual"`). For example to initialize all the intercept terms at 0.5 and all of the regression terms corresponding to the restricted covariate space in the above example at one, run

```
initValue = list( "init01" = c( 0.5, rep( 1, 6 )),
                  "init10" = c( 0.5, rep( 1, 4 )),
                  "init00" = c( 0.5, rep( 1, 4 )),
                  "init11" = c( 0.5, rep( 1, 3 )) )
```

and set the `initvalue` argument in the `HMMbvs` function to `initValue`. Here, `initvalue` accepts a list of size four, and the dimension of each element in the list must match the dimension of the corresponding element in `tcova` and `ecova` plus one, where the first value of each element corresponds to the initialization of the intercept.

In this example, we set the number of MCMC iterations to run as 25,000 (`iter = 25000`). The number of iterations can be determined based on the convergence of the model. The default values for the prior variance and the proposal variance are five (`v1 = 5`) and one (`v2 = 1`), respectively. The prior variance indicates how diffuse the prior distribution for active regression coefficients is. For example, `v1 = 5` places a 95% prior probability of included regression coefficients between a hazards/odds ratio of 0.01 and 80. The default hyperparameters for the prior inclusion indicators are `a = 1, b = 9`, where $\frac{a}{a+b}$ reflects the prior belief regarding the proprotion of covariates active in the model. Due to the potentially large parameter space of the model, the MCMC chains may require more memory than available. For example, in our simulation study (150 subjects with 30 transitions for each subject and 25,000 MCMC iterations), the hidden states alone would require about 1 Gb of memory. For a quick estimate of the memory required for model output, a dataset with 100,000 cells would require 0.8 Mb of space. To help manage memory storage, the user can control the amount of thinning with the `thin` and `thin_hidden` arguments. By default, the regression coefficient and hidden state MCMC samples are thinned to every 10^{th} iteration, `thin = 10` and `thin_hidden = 10`, respectively. Finally, to deal with the potential label switching issue, we can use `base01larger` and/or `base00larger` to force one of the intercept terms in the transition and/or the emission to be larger than the other. By default their values are set to `-1`, meaning that there are no constraints enforced. We can set `base01larger=1` (or `0`) such that $\lambda_0 > \mu_0$ (or $<$) in the model. Similarly, `base00larger=1` (or `0`) implies $\beta_{0,N} > \beta_{0,S}$ (or $<$).

To run an MSM, set `model = "msm"` and the emission-related arguments (`ecova, eforce`) to their default values as follows:

```
msmout = HMMbvs_R( data = df1$DATA,
           tcova = c( "tVar1", "tVar2", "tVar3", "tVar4", "tVar5", "tVar6", "tVar7", "tVar8",
                      "tVar9", "tVar10", "tVar11", "tVar12", "tVar13", "tVar14", "tVar15" ),
           ecova = NULL, tforce = NULL, eforce = NULL, standardize = NULL, model = "MSM",
           init = "baseline", initvalue = NULL, iter = 5000, v1 = 5, v2 = 1, a = 1, b = 9,
           thin = 10, thin_hidden = 10 )
```

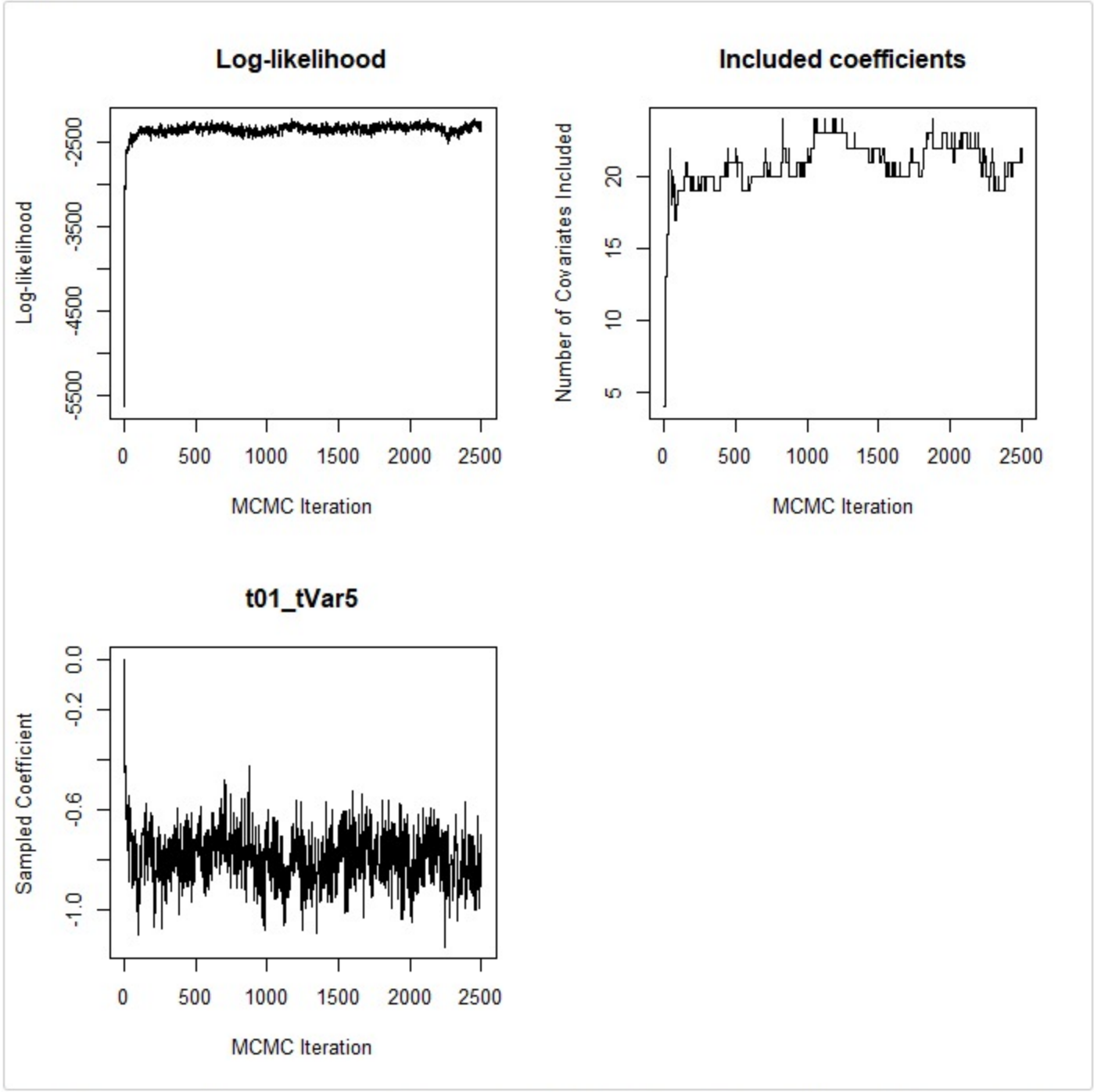
From our experience, MSMs require less iterations for convergence. Thus, in this example we reduce the number of iterations to 5,000. Additionally, note that the `thin_hidden` will be ignored when the function is specified for MSMs.

Convergence and Model Fit

Prior to analyzing the model output, we recommend assessing convergence and goodness-of-fit for the model. There are numerous ways to assess the covergence of the MCMC algorithm. A standard approach for visually assessing model convergence is to generate trace plots for each coefficient in the model. To make trace plots for the posterior samples of each regression coefficient, we take advantage of the `coda` package in R. For the large model spaces typically found in variable selection research, an alternative way to visually assess overall convergence of the model is to plot the joint log-likelihood of observed and hidden states and/or the number of covariates active in the model as a function of MCMC iterations. Convergence is determined once these plots have stabilized across MCMC iterations. Our package contains a function to generate each of these plots,

```
convergence( output = hmmout, file = "convergence.pdf")
```

which requires the output from the `HMMbvs()` function, (`output = hmmout`), and the name of the pdf file, (`file = "convergence.pdf"`), where the plots will be saved in the current working directory in R. By adjusting the input for the `file` argument, users can also save this pdf file elsewhere. For example, set `file = "<path>/convergence.pdf"`. For more advanced usage of the `file` argument, please refer to the documentation of the `pdf()` function in R.



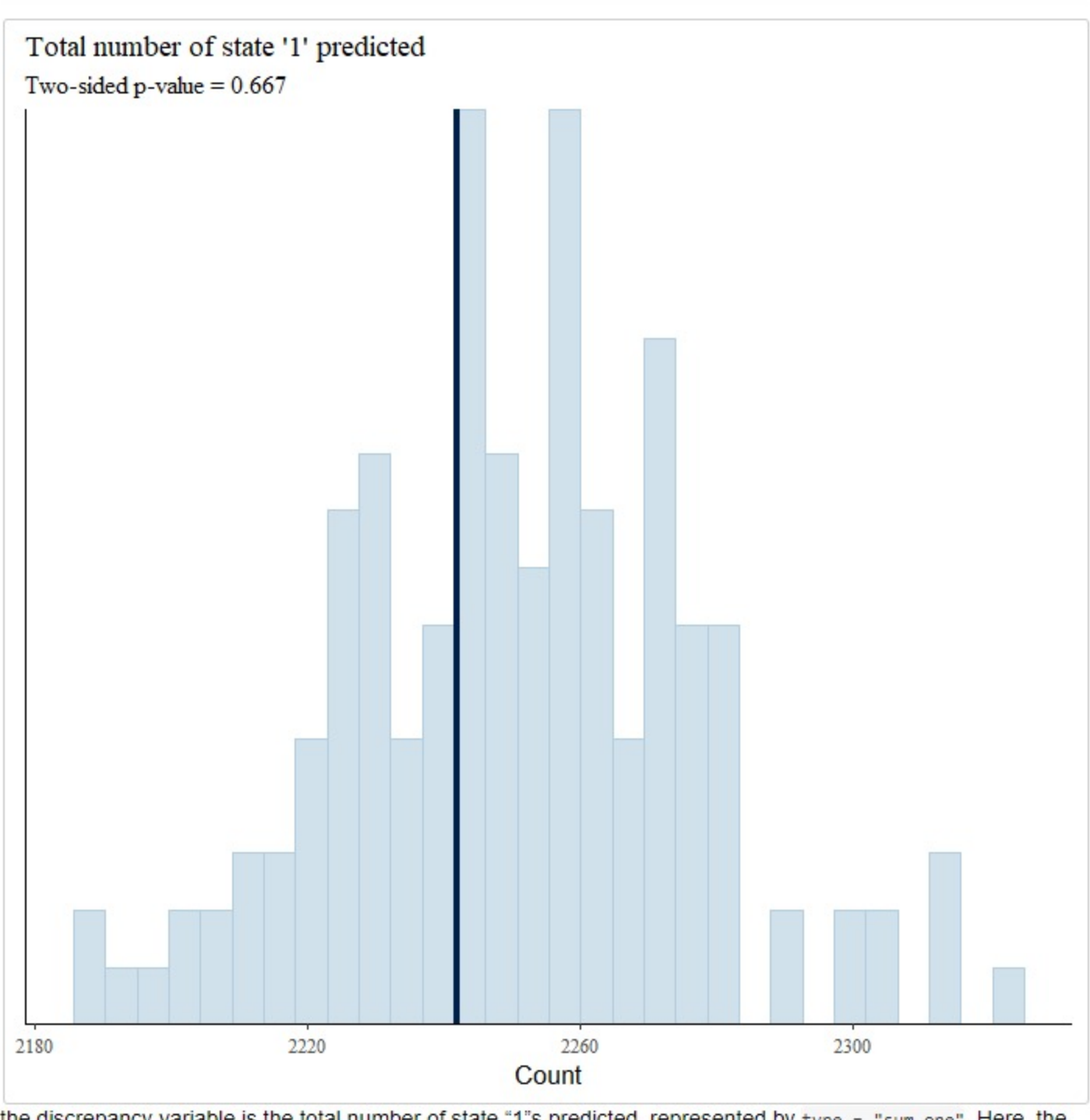
The first two pages of the generated pdf document contain plots of the joint log-likelihood and the total number of covariates included across all MCMC iterations. These two plots can be used to determine if the number of MCMC iterations is sufficient (i.e., the plots have stabilized) and how many iterations should be discarded prior to convergence (also known as the burn-in). In our example, the MCMC iterations are thinned to every 10^{th} iteration, and both of the plots seem to converge by the $1{,}000^{th}$ iteration. Thus, we will use a burn-in of 1,000 iterations, using the remaining 1,500 for inference.

The subsequent pages in the pdf correspond to trace plots for each of the regression coefficients. Trace plot titles are formatted as *process_variable* to indicate the transition or emissions direction and the name of the variable. The bottom left figure above shows an example of one of the trace plots produced. The title indicates that it is the trace plot for covariate "tVar6" modeling the transition from 0 to 1.

To generate plots to assess the convergence of an MSM, change the `output` argument to `output = msmout` and rename the pdf output file accordingly.

To assess goodness-of-fit for Bayesian models with discrete outcomes, Gelman et al. (2000) recommend posterior predictive checks (PPCs) that compare replicated data sets from the posterior predictive distribution of the model to the observed data. For each replicated data set, a discrepancy variable (e.g., the total number of observed smoking states), is compared to the observed data. In theory, the model fit these data well, if the discrepancy variable calculated on the observed data appears to of been generated by the model. To generate PPCs, we take advantage of the `Bayesplot` package in R. In this example using the `ppc` function,

```
ypost = ppc( output = hmmout, file = "sum_one.pdf", type = "sum_one", burnin = 1000, postsample = NULL)
```



the discrepancy variable is the total number of state "1"s predicted, represented by `type = "sum_one"`. Here, the `output` and `file` arguments are defined similarly to the `convergence()` function. The `burnin` argument indicates the number iterations to discard based on the convergence assessment. The `type` argument accepts one (or more) of the following in vector form: `"sum_one", "avg", "var", "tran01", "tran10", "tran00", "tran11"`, representing the discrepancy functions for the total number of state "1"s predicted, the mean and variance of average proportion of state "1"s predicted per subject, and the total number of transitions predicted for $0 \rightarrow 1$, $1 \rightarrow 0$, $0 \rightarrow 0$ and $1 \rightarrow 1$. In this tutorial, we present results from `type = "sum_one"` (Figure above). The plot generated by this function is a histogram of the total number of state "1"s generated from the posterior predictive distribution calculated with the parameter values at each MCMC iteration. The dark blue line represents the total number of state "1"s observed in the dataset. This plot indicates good model fit, as the 2-sided p-value for rejecting the goodness-of-fit is over 0.8. Sampling the posterior predictive distribution of the discrepancy variable can take a long time. Hence we make the `ppc()` function output the posterior sampling data matrix, in this example, to `ypost`. If we need to make different types of PPC plots using the same output object, simply put this posterior sampling data matrix to the `postsample` argument, e.g. (`postsample = ypost`), that will save some computational time. For example, we can run the following to generate the plots for all other discrepancy variables listed:

```
ppc( output = hmmout, file = "ppc_plot.pdf", type = c("avg", "var", "tran01",
    "tran10", "tran00", "tran11"), burnin = 1000, postsample = ypost)
```

Currently the `ppc()` function only works for the HMMs.

Posterior Inference

Having checked convergence and fit of the model, we now demonstrate how to draw inference from the MCMC output for variable selection using the following function

```
out = selection( output = hmmout, burnin = 1000, cred = 0.95, threshold = 0.5, trueGamma = NULL,
                 plotting = TRUE, file = "mppi.pdf")
```

The `selection` function requires the model output (`output = hmmout`) and the number of MCMC iterations to treat as burn in, (`burnin = 1000`). The `out` object contains the marginal posterior probability of inclusion (MPPI) for each of the corresponding regression coefficients, as well as a dataframe which includes the posterior mean of the regression coefficients and their credible intervals. By default, a 95% credible intervals is generated (`cred = 0.95`), and an MPPI threshold of 0.5 (`threshold = 0.5`) is used to determine inclusion in the model. The MPPIs and tables for all of the regression coefficients in the full model can be extracted using the following code,

```
mppi = out$mppi
table_full = out$full_table
```

To extract a table with only the selected regression coefficients, run

```
table_selected = out$table
```

The `mppi` object is a numeric vector containing the MPPIs for all the terms in the model. The first five rows of the `table_full` object, extracted using the `head()` function in R are

```
head(table_full)
#>              MPPI    Est CI_2.5 CI_97.5
#> baseline01 1.000 -0.165 -0.725   0.365
#> t01_tVar1  0.751 -0.644 -0.986  -0.322
#> t01_tVar2  0.007 -0.132 -0.317   0.162
#> t01_tVar3  0.003 -0.221 -0.302  -0.157
#> t01_tVar4  0.000    NaN     NA      NA
#> t01_tVar5  1.000 -0.808 -0.996  -0.614
```

In this table, `Terms` indicates the name of the corresponding covariate as well as its location in the model, similar to the format used to label the trace plots. Intercept terms' estimates, which can be interpreted as baseline log odds/hazards, are labeled as "baseline*xy*", where "*xy*" takes on values of either 01, 10, 00, or 11 indicating the direction of transition or emissions the term represents. `MPPI` indicates the marginal posterior probability of inclusion for the corresponding term, empirically estimated as the average of the inclusion indicators across MCMC iterations. The posterior mean estimates, `Est`, are interpreted as log hazards or log odds ratios, depending on if the covariate is included in the transitions or emissions part of the model. Note that an MPPI equal to one indicates that the covariate (or intercept term) was forced into the model as specified by the user, or after burn-in, the algorithm never excluded the term from the model. Also, the posterior mean estimates and the credible intervals are calculated using samples with corresponding inclusion indicator equal to 1. Hence if one term is never included (e.g. `` `t01\_tVar3'' in our example) in the model, both the estimated and the credible intervals will be shown as NAs. The``table_selected` dataframe is structured similarly, but only contains terms that were selected based on the given MPPI threshold.

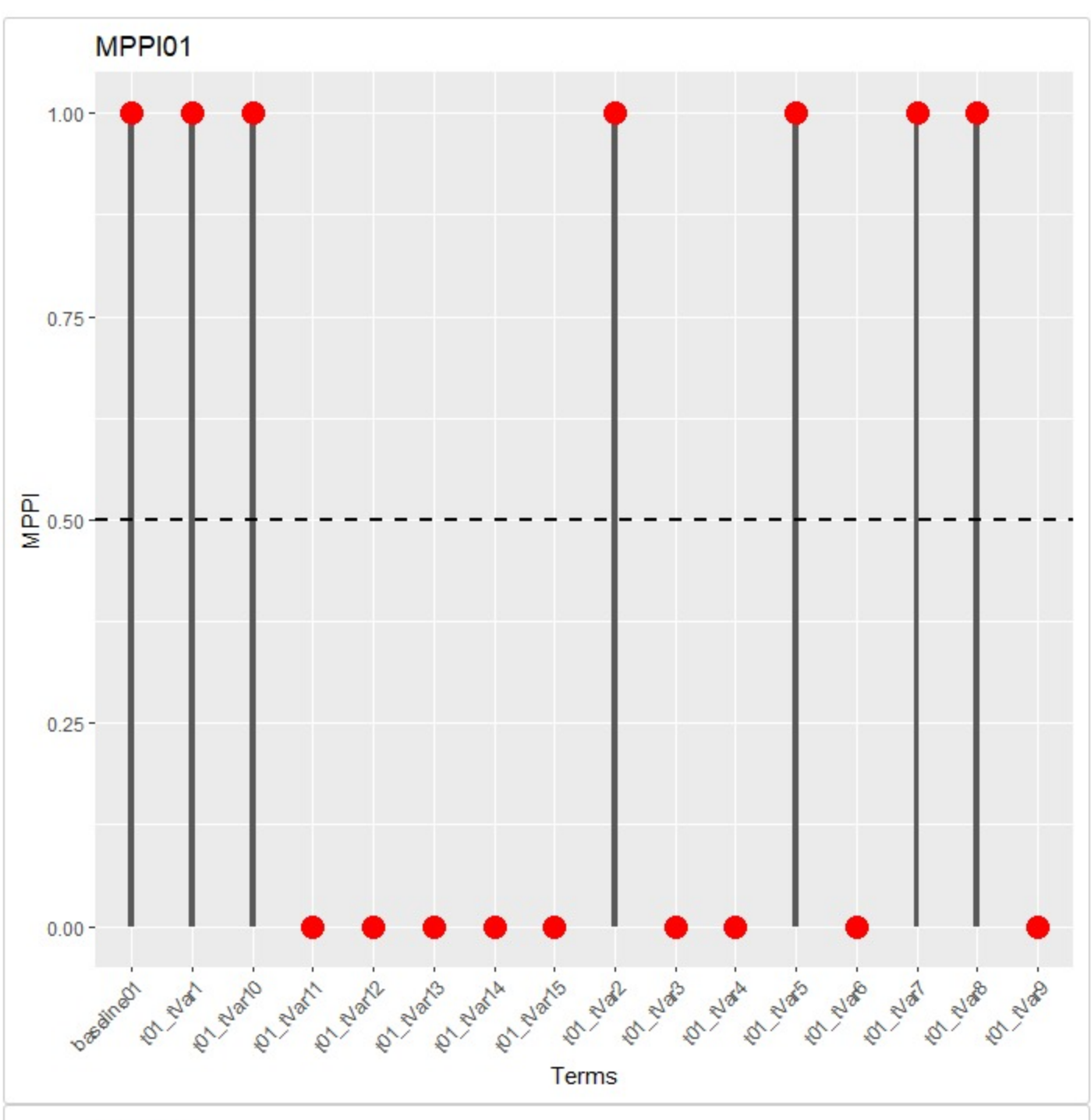
If the `plotting` argument in the `selection()` function is set to `TRUE`, a plot of each term's corresponding MPPI for each part of the model (i.e., transition $0 \rightarrow 1$ and $1 \rightarrow 0$, as well as emissions $0 \rightarrow 0$ and $1 \rightarrow 1$ is generated and saved as a pdf (`file = "mppi.pdf"`). For example in Fig., "MPPI01" indicates the transition from 0 to 1 and, "MPPI00" indicates the emission from 0. A horizontal dotted line at the given MPPI threshold is plotted against the estimated MPPIs to help users visualize which terms are included and excluded in the model.

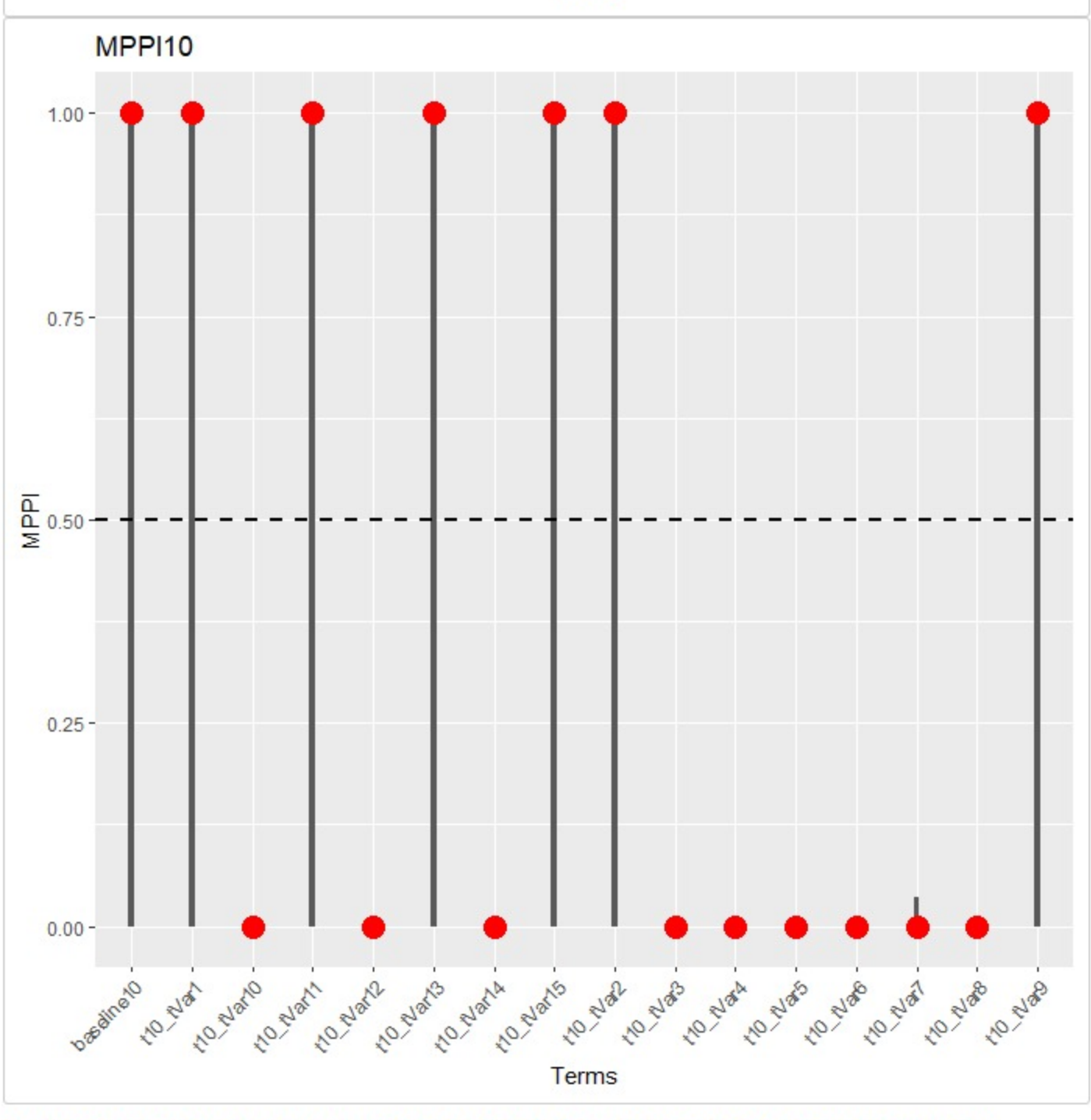
In simulation, the true associations are known and can be compared to the selection results. To visualize the selection performance of the model in simulation, the `selection` function accepts a dataframe of the true associations (i.e., inclusion indicators equal to 1) and plots them as red dots against the corresponding MPPI, as seen in Fig.. To extract the true associations from the simulated data and include them in the MPPI plots, run

```
truegamma = findtruegamma( data = df, output=hmmout )
out = selection( output = hmmout, burnin = 1000, cred=0.95, threshold = 0.5, trueGamma = truegamma,
                 plotting = TRUE, file = "mppi.pdf")
```

For an MSM model, the process is similar. For example, the transition estimate tables and corresponding MPPI plots for an MSM model can be generated with the following code.

```
truegamma = findtruegamma( data = df1, output=msmout )
out = selection( output = msmout, burnin = 250, cred=0.95, threshold = 0.5, trueGamma = truegamma,
                 plotting = TRUE, file = NULL)
```





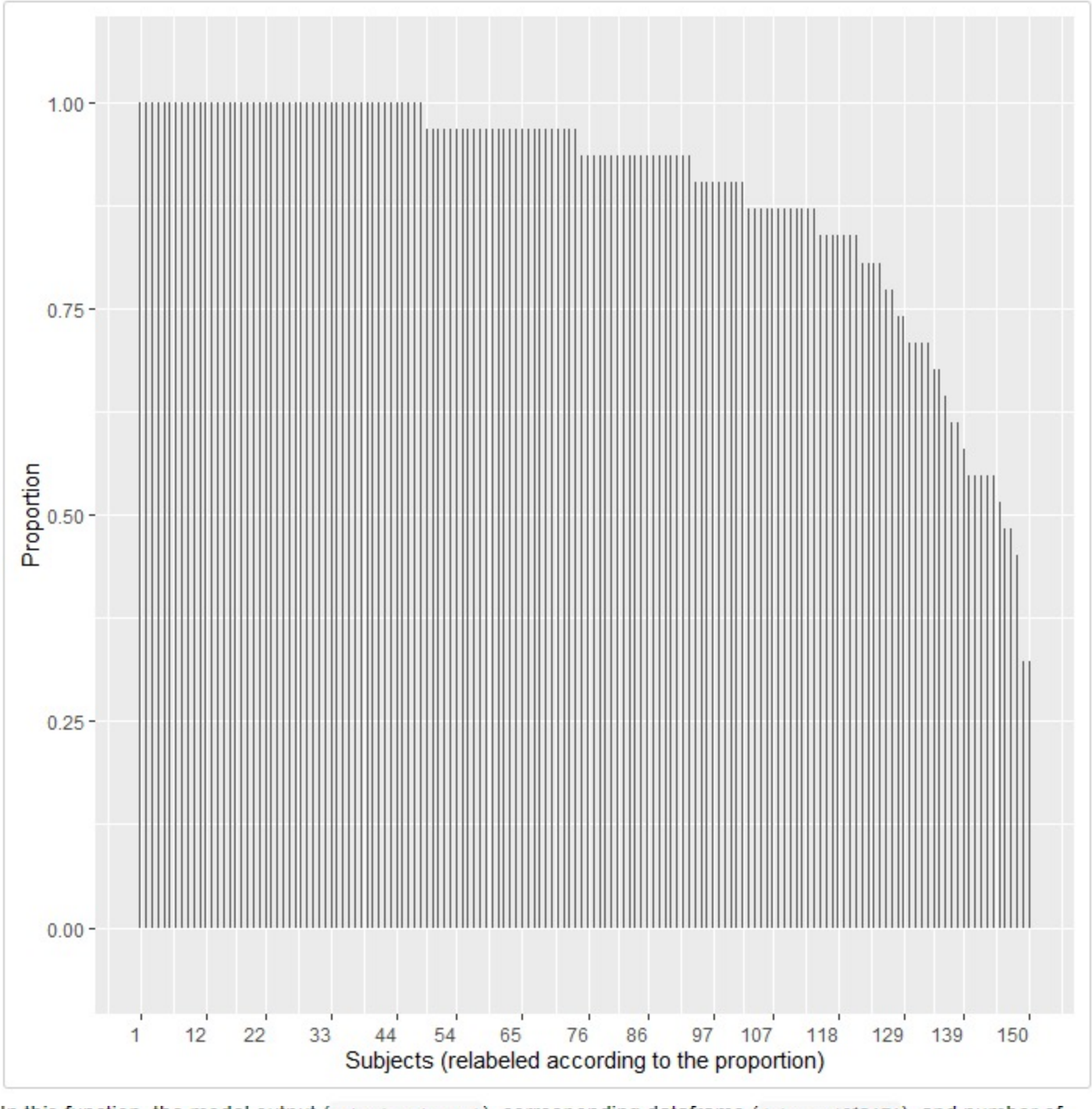
An alterative approach for determining a covariate's inclusion in the model is based on a Bayesian false discovery rate (BFDR) threshold, which controls for multiplicity. A BFDR is the Bayesian analogy for calculating the expectation of the type-I error rate. Hence, different inclusion thresholds map to different BFDRs. In `HMMbvs`, we provide a function that calculates thresholds corresponding to BFDRs from 0.05 to 0.5. Inputting the MPPIs estimated by the model in our example generates the following table

```
bfdr_tab = bfdr_wrap( mppi )
print(bfdr_tab)
#>        FDR   Threshold Selected
#>  [1,] 0.05 0.751333333       20
#>  [2,] 0.10 0.427333333       22
#>  [3,] 0.15 0.068666667       24
#>  [4,] 0.20 0.016000000       25
#>  [5,] 0.25 0.012000000       27
#>  [6,] 0.30 0.007333333       29
#>  [7,] 0.35 0.004666667       32
#>  [8,] 0.40 0.002666667       34
#>  [9,] 0.45 0.002000000       37
#> [10,] 0.50 0.000000000       40
```

where `FDR`, `Threshold`, and `Selected` represent the BFDR, corresponding threshold, and the number of terms selected at the given threshold. In this example, the median model approach obtained a BFDR of 0.10. For interpretation, we would expect 10% of the 22 selected terms to be false positives.

In order to visualize the estimated hidden states relative to the observed states, a plot of the proportion of matched states for the sample is generated using the following code

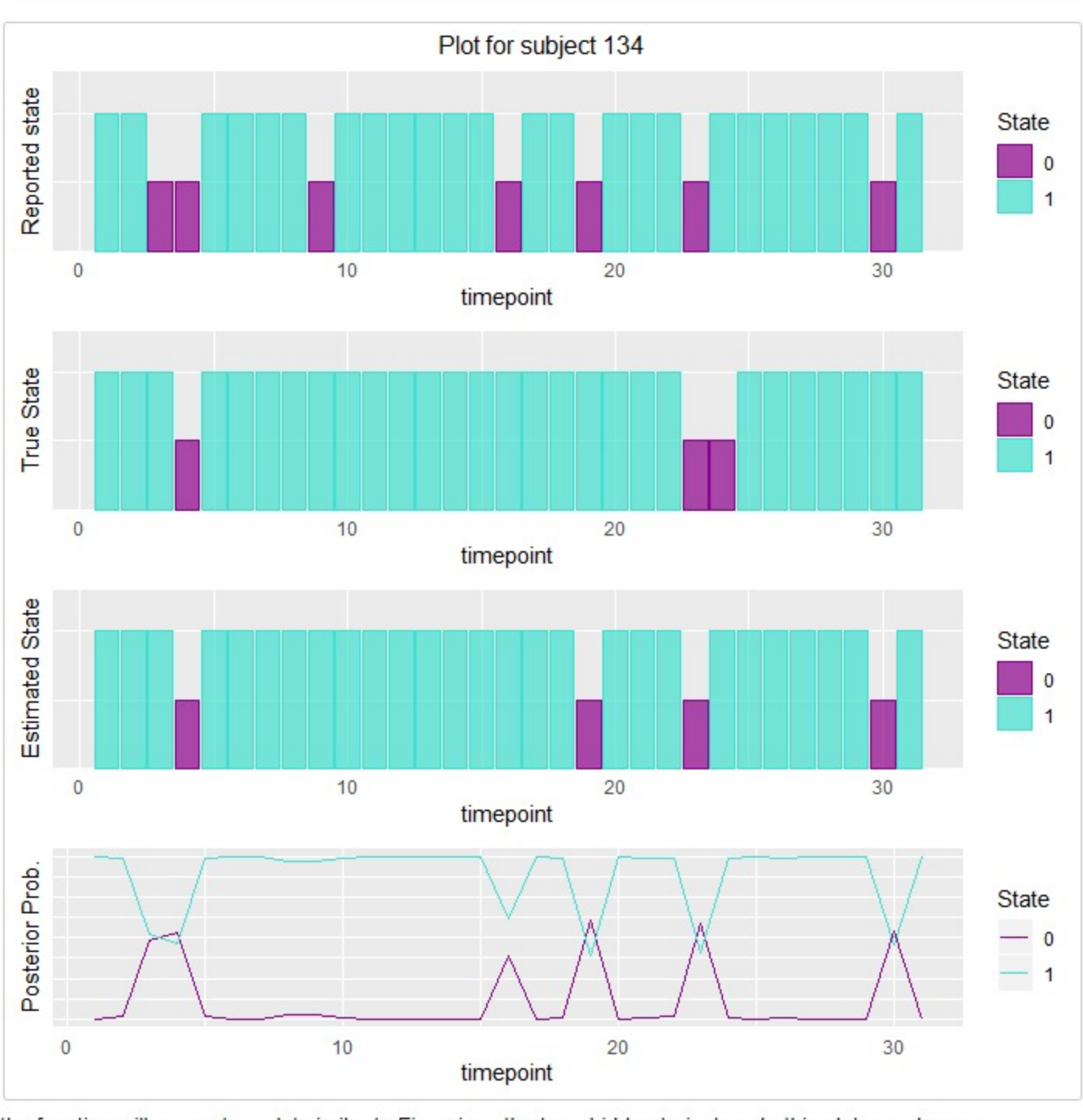
```
plot_hidden(output=hmmout, data=df$DATA, burnin=1000, subject="all", true_hidden=df$STATE)
```



In this function, the model output (`output = hmmout`), corresponding dataframe (`data = df$DATA`), and number of MCMC iterations to treat as burn-in (`burnin=1000`) are required to generate the plots. Users can additionally set the `true_hidden` argument to a vector consisting of the true hidden states (e.g., `truth = df$hidden`) when available. By doing so, a plot of the proportion of true hidden states matching the corresponding reported states for all individuals (`subject = "all"`) is generated. Note that the proportions are sorted in descending order and hence the subject indices are relabeled and different from the original indices. This plot can be used to visualize the overall proportion of mismatched states in the sample. In the absence of true hidden state information (i.e., `truth = NULL`), a similar figure will be generated which presents the proportion of hidden states estimated by our model that match the reported states for all individuals.

To visualize the relation between the estimated hidden and reported state sequence at the individual level, simply change the `subject` argument. For example, by setting `subject = 134` as the follows:

```
plot_hidden(output=hmmout, data=df$DATA, burnin=1000, subject=134, true_hidden=df$STATE)
```



the function will generate a plot similar to Fig., given the true, hidden trajectory. In this plot, we observe misreported smoking behaviors at time points 3, 9, 16, 19, 24 and 30. By using our model, we successfully caught 50% of the reporting errors. If the true, hidden states are unknown, setting `true_hidden = NULL` will suppress the plot of the true smoking trajectories. This function also works for HMMs only since the MSM model doesn't have a hidden layer and the plotting will not be very meaningful.