Vignette for `adaptBayes`

Philip S. Boonstra

October 31, 2019

Introduction

This vignette presents a step-by-step approach for using the `R` functions `glm_standard()`, `glm_nab()`, and `glm_sab()` in the `adaptBayes` package.

First, install and load the `adaptBayes` package and other necessary packages:

```
if(!require(adaptBayes)) {
  library(devtools)
  # if installation is necessary, compiling everything will take a few minutes
  install_github('umich-biostatistics/adaptBayes')
}

library(mice);library(Hmisc);library(MASS);
library(rstan);library(Matrix);library(mnormt);
library(tidyverse);

# some recommended options from the STAN development team
options(mc.cores = parallel::detectCores());
rstan_options(auto_write = TRUE);
```

Also, source the script that contains the data-simulating function `draw_data()` and the function `solve_for_hiershrink_scale()`, which is used to solve for the scale parameter.

```
source("functions_simulation.R"); # For access to the draw_data() function
```

Draw data

```
## Draw Data ====################################################
# Choose your own values if desired
set.seed(1);
n_hist = 500;
n_curr = 100;
n_new = 1e3;
# this is different from the misspecified marginal prevalence;
# see Remark 3 in the manuscript
true_mu_hist = 0;
true_mu_curr = -2.5;
# original betas common to both analyses:
true_betas_orig = c(2,-2,1,-1);
# augmented betas exclusive to current analysis:
true_betas_aug = c(-1,-1,0.5,0.5,-0.25,-0.25);
covariate_args = list(x_correlation = 0.2,
                      x_orig_binom = 1:length(true_betas_orig),
                      x_aug_binom = 1:length(true_betas_aug));
complete_dat = draw_data(n_hist = n_hist,
                         n_curr = n_curr,
                         n_new = n_new,
                         true_mu_hist = true_mu_hist,
```

```
                          true_mu_curr = true_mu_curr,
                          true_betas_orig = true_betas_orig,
                          true_betas_aug = true_betas_aug,
                          covariate_args = covariate_args);
orig_covariates = paste0("orig",1:length(true_betas_orig));
aug_covariates = paste0("aug",1:length(true_betas_aug));
y_hist = complete_dat$y_hist;
y_curr = complete_dat$y_curr;
x_hist_orig = as.matrix(complete_dat$x_hist_orig);
x_curr_orig = as.matrix(complete_dat$x_curr_orig);
colnames(x_hist_orig) =
  colnames(x_curr_orig) = orig_covariates;
x_curr_aug = as.matrix(complete_dat$x_curr_aug);
# 'x_hist_aug' is essentially missing here, since those data
# were not collected in the historical analysis
colnames(x_curr_aug) = aug_covariates;
p = length(true_betas_orig);
q = length(true_betas_aug);
```

Methods to fit

'Historical' is a horseshoe prior applied only to the historical data. It is a method in and of itself as well as the prior analysis that will be provided to the NAB / SAB methods.

'Standard' is a horseshoe prior applied to the current data. It is presumably what would be done in the absence of any knowledge about the historical analysis.

'NAB' and 'SAB' are the naive and sensible adaptive Bayesian priors, respectively.

Here the hyperparameters for ϕ are also described. The truncation to the interval [0,1] is always assumed and not necessary to specify.

```
##Methods to fit====##########################################
# Each element of this list will be crossed with each adaptive
# prior.
phi_params = list("Agnostic" = c(mean = 0.5, sd = 2.5),
                  "Optimist" = c(mean = 1, sd = 0.25));

base_meth_names = c("Historical",
                    "Standard",
                    "NAB",
                    "SAB");
expanded_meth_names = c("Historical",
                        "Standard",
                        paste0("NAB",names(phi_params)),
                        paste0("SAB",names(phi_params)));
```

Model hyperparameters

Specify the hyperparameters, including deriving values of c using the function `solve_for_hiershrink_scale`

```
## Model hyperparameters====################
local_dof = 1;
global_dof = 1;
slab_precision = (1/15)^2; # 'd' in the paper
nab_augmented_scale = 0.05;# 'tilde_c' in the paper
```

```r
sab_imputes_list = list(c(1,100)); # Section S1 supplement
stopifnot(class(sab_imputes_list) == "list");
sab_num_imputes_each = unlist(lapply(sab_imputes_list,diff)) + 1;
max_sab_index = max(unlist(lapply(sab_imputes_list,max)));
min_sab_index = min(unlist(lapply(sab_imputes_list,min)));
store_hierarchical_scales =
  #prior effective number of original parameters = mean(rowSums(1-kappa[orig]))
  prior_eff =
  vector("list",length(base_meth_names));
names(store_hierarchical_scales) =
  names(prior_eff) =
  base_meth_names;
power_prop_nonzero_prior = 1/3;

# 'c' for Historical
foo = solve_for_hiershrink_scale(target_mean1 = -0.5 + p ^ power_prop_nonzero_prior,
                                 target_mean2 = NA,
                                 npar1 = p,
                                 npar2 = 0,
                                 local_dof = local_dof,
                                 regional_dof = -Inf,
                                 global_dof = global_dof,
                                 n = n_hist,
                                 sigma = 2,
                                 n_sim = round(2e6/(p + q)),
                                 slab_precision = slab_precision);
store_hierarchical_scales$Historical = foo$scale1;
prior_eff$Historical = foo$prior_num1;
rm(foo);

# 'c' for the Standard, NAB, and SAB models
foo = solve_for_hiershrink_scale(target_mean1 = -0.5 + (p + q) ^ power_prop_nonzero_prior,
                                 target_mean2 = NA,
                                 npar1 = p + q,
                                 npar2 = 0,
                                 local_dof = local_dof,
                                 regional_dof = -Inf,
                                 global_dof = global_dof,
                                 n = n_curr,
                                 sigma = 2,
                                 n_sim = round(2e6/(p + q)),
                                 slab_precision = slab_precision);
store_hierarchical_scales$Standard =
  store_hierarchical_scales$NAB =
  store_hierarchical_scales$SAB =
  foo$scale1;
prior_eff$Standard =
  prior_eff$NAB =
  prior_eff$SAB =
  foo$prior_num1;
rm(foo);
#
store_hierarchical_scales$NAB_aug_tilde = nab_augmented_scale;
```

Model hyperparameters

```
## MC Params ====#############################################
mc_warmup = 1e3;
mc_iter_after_warmup = 1e3;
only_prior = 0;# set to 1 if you only want to sample from prior
mc_chains = 2;
mc_thin = 1;
mc_stepsize = 0.1;
mc_adapt_delta_relaxed = 0.99;
mc_adapt_delta_strict = 0.999;
mc_max_treedepth = 15;
ntries_per_iter = 2;
```

Methods: Historical

```
## Historical ====##########################################
#Historical analysis only
curr_method = "Historical";
y = y_hist;
x_standardized = x_hist_orig;
beta_orig_scale = store_hierarchical_scales[[curr_method]];
beta_aug_scale = store_hierarchical_scales[[curr_method]];
```

The values `p` and `q` should add up to be equal to the number of columns in `x_standardized`. It is assumed that the first `p` columns correspond to the original covariates, and the second `q` columns correspond to the augmented covariates. For `glm_standard`, the only difference is that you can specify different scale hyperparameters to be applied to the original and augmented regression coefficients.

```
foo = glm_standard(y = y,
                   x_standardized = x_standardized,
                   p = p,
                   q = 0,
                   beta_orig_scale = beta_orig_scale,
                   beta_aug_scale = beta_aug_scale,
                   local_dof = local_dof,
                   global_dof = global_dof,
                   slab_precision = slab_precision,
                   intercept_offset = NULL,
                   only_prior = only_prior,
                   mc_warmup = mc_warmup,
                   mc_iter_after_warmup = mc_iter_after_warmup,
                   mc_chains = mc_chains,
                   mc_thin = mc_thin,
                   mc_stepsize = mc_stepsize,
                   mc_adapt_delta = mc_adapt_delta_relaxed,
                   mc_max_treedepth = mc_max_treedepth,
                   ntries = ntries_per_iter);

##Keep copy of values;
assign(paste0("beta0_",curr_method),foo$hist_beta0);
assign(paste0("beta_",curr_method),foo$curr_beta);

#See what else is stored
names(foo);
```

```
## [1] "accepted_divergences" "max_divergences"      "max_rhat"
## [4] "hist_beta0"           "curr_beta0"           "curr_beta"
## [7] "theta_orig"           "theta_aug"
```

```
#Garbage cleanup
rm(foo, curr_method, y, x_standardized, beta_orig_scale, beta_aug_scale);
```

Methods: Standard

```
## Standard ====##############################################
#Standard analysis of current data, ignoring historical model
curr_method = "Standard";
y = y_curr;
x_standardized = cbind(x_curr_orig,x_curr_aug);
beta_orig_scale = store_hierarchical_scales[[curr_method]];
beta_aug_scale = store_hierarchical_scales[[curr_method]];
```

As in the previous model, the values `p` and `q` need to add up to be equal to the number of columns in `x_standardized`. But note now that `x_standardized` has more columns and contains a different set of observations.

```
foo = glm_standard(y = y,
                   x_standardized = x_standardized,
                   p = p,
                   q = q,
                   beta_orig_scale = beta_orig_scale,
                   beta_aug_scale = beta_aug_scale,
                   local_dof = local_dof,
                   global_dof = global_dof,
                   slab_precision = slab_precision,
                   intercept_offset = NULL,
                   only_prior = only_prior,
                   mc_warmup = mc_warmup,
                   mc_iter_after_warmup = mc_iter_after_warmup,
                   mc_chains = mc_chains,
                   mc_thin = mc_thin,
                   mc_stepsize = mc_stepsize,
                   mc_adapt_delta = mc_adapt_delta_relaxed,
                   mc_max_treedepth = mc_max_treedepth,
                   ntries = ntries_per_iter);

##Keep copy of values;
assign(paste0("beta0_",curr_method),foo$hist_beta0);
assign(paste0("beta_",curr_method),foo$curr_beta);

#See what else is stored
names(foo);
```

```
## [1] "accepted_divergences" "max_divergences"      "max_rhat"
## [4] "hist_beta0"           "curr_beta0"           "curr_beta"
## [7] "theta_orig"           "theta_aug"
```

```
#Garbage cleanup
rm(foo, curr_method, y, x_standardized, beta_orig_scale, beta_aug_scale);
```

Methods: NAB

```
## NAB ====##########################################################
#Naive Adaptive Bayes: apply Historical analysis directly as a prior on beta_orig.

curr_base_method = "NAB";
y = y_curr;
x_standardized = cbind(x_curr_orig,x_curr_aug);
beta_orig_scale = store_hierarchical_scales[[curr_base_method]];
beta_aug_scale = store_hierarchical_scales[[curr_base_method]];
beta_aug_scale_tilde = store_hierarchical_scales[[paste0(curr_base_method,"_aug_tilde")]];

###
#These will all be needed for SAB also
alpha_prior_mean = colMeans(beta_Historical);
alpha_prior_cov = var(beta_Historical);
scale_to_variance225 = diag(alpha_prior_cov) / 225;
eigendecomp_hist_var = eigen(alpha_prior_cov);
###

prior_type = names(phi_params)[1];

for(prior_type in names(phi_params)) {

  curr_method = paste0(curr_base_method,prior_type);
  phi_mean = eval(phi_params[[prior_type]][["mean"]]);
  phi_sd = eval(phi_params[[prior_type]][["sd"]]);

  foo = glm_nab(y = y,
                x_standardized = x_standardized,
                alpha_prior_mean = alpha_prior_mean,
                alpha_prior_cov = alpha_prior_cov,
                phi_mean = phi_mean,
                phi_sd = phi_sd,
                beta_orig_scale = beta_orig_scale,
                beta_aug_scale = beta_aug_scale,
                beta_aug_scale_tilde = beta_aug_scale_tilde,
                local_dof = local_dof,
                global_dof = global_dof,
                slab_precision = slab_precision,
                only_prior = only_prior,
                mc_warmup = mc_warmup,
                mc_iter_after_warmup = mc_iter_after_warmup,
                mc_chains = mc_chains,
                mc_thin = mc_thin,
                mc_stepsize = mc_stepsize,
                mc_adapt_delta = mc_adapt_delta_strict,
                mc_max_treedepth = mc_max_treedepth,
                ntries = ntries_per_iter,
                eigendecomp_hist_var = eigendecomp_hist_var,
                scale_to_variance225 = scale_to_variance225);

  ##Keep copy of values;
  assign(paste0("beta0_",curr_method),foo$hist_beta0);
  assign(paste0("beta_",curr_method),foo$curr_beta);
  assign(paste0("phi_",curr_method),foo$phi);
```

```r
  #See what else is stored
  names(foo);

}

rm(foo, curr_method, y, x_standardized, beta_orig_scale, beta_aug_scale, beta_aug_scale_tilde, prior_type)
```

Methods: SAB

```r
## SAB ====#######################################################
#Sensible Adaptive Bayes: apply Historical analysis as a prior on beta_orig + projection%*%beta_aug
curr_base_method = "SAB";
y = y_curr;
x_standardized = cbind(x_curr_orig,x_curr_aug);
beta_orig_scale = store_hierarchical_scales[[curr_base_method]];
beta_aug_scale = store_hierarchical_scales[[curr_base_method]];
```

This function creates the projection matrix P in Equation (3.8) of the manuscript

```r
aug_projection = create_projection(x_curr_orig = x_curr_orig,
                                   x_curr_aug = x_curr_aug,
                                   eigenvec_hist_var = t(eigendecomp_hist_var$vectors),
                                   imputes_list = sab_imputes_list);
```

```
## Loading required package: magrittr

##
## Attaching package: 'magrittr'

## The following object is masked from 'package:purrr':
##
##     set_names

## The following object is masked from 'package:tidyr':
##
##     extract

## The following object is masked from 'package:rstan':
##
##     extract
```

```r
prior_type = names(phi_params)[1];

for(prior_type in names(phi_params)) {

  curr_method = paste0(curr_base_method,prior_type);
  phi_mean = eval(phi_params[[prior_type]][["mean"]]);
  phi_sd = eval(phi_params[[prior_type]][["sd"]]);

  foo = glm_sab(y = y,
                x_standardized = x_standardized,
                alpha_prior_mean = alpha_prior_mean,
                alpha_prior_cov = alpha_prior_cov,
                aug_projection = aug_projection[[1]],
                phi_mean = phi_mean,
                phi_sd = phi_sd,
                beta_orig_scale = beta_orig_scale,
                beta_aug_scale = beta_aug_scale,
```

```
                local_dof = local_dof,
                global_dof = global_dof,
                slab_precision = slab_precision,
                only_prior = only_prior,
                mc_warmup = mc_warmup,
                mc_iter_after_warmup = mc_iter_after_warmup,
                mc_chains = mc_chains,
                mc_thin = mc_thin,
                mc_stepsize = mc_stepsize,
                mc_adapt_delta = mc_adapt_delta_strict,
                mc_max_treedepth = mc_max_treedepth,
                ntries = ntries_per_iter,
                eigendecomp_hist_var = eigendecomp_hist_var,
                scale_to_variance225 = scale_to_variance225);


  ##Keep copy of values;
  assign(paste0("beta0_",curr_method),foo$hist_beta0);
  assign(paste0("beta_",curr_method),foo$curr_beta);
  assign(paste0("phi_",curr_method),foo$phi);
  #See what else is stored
  names(foo);
}

rm(foo, curr_method, y, x_standardized, beta_orig_scale, beta_aug_scale, prior_type);
rm(aug_projection, alpha_prior_mean, alpha_prior_cov, scale_to_variance225, eigendecomp_hist_var);
```

Results

```
## Results ====#####################################################

# Posterior mean
colMeans(beta_Standard);
```

```
##  [1]  1.55037 -1.77180  0.51693 -0.15743 -0.19236 -0.61620  0.28307
##  [8]  0.13598  0.00629 -0.12599
```

```
colMeans(beta_NABAgnostic);
```

```
##  [1]  1.41468 -1.59108  0.68502 -0.59448 -0.04733 -0.36156  0.11458
##  [8]  0.03384  0.00354 -0.04656
```

```
colMeans(beta_NABOptimist);
```

```
##  [1]  1.40578 -1.58686  0.68654 -0.60317 -0.03917 -0.33323  0.10189
##  [8]  0.03152  0.00172 -0.04044
```

```
colMeans(beta_SABAgnostic);
```

```
##  [1]  1.33953 -1.52072  0.84295 -0.59815 -0.22958 -0.29944  0.20200
##  [8]  0.05529 -0.00776 -0.12761
```

```
colMeans(beta_SABOptimist);
```

```
##  [1]  1.33028 -1.51865  0.83843 -0.61460 -0.20351 -0.25464  0.18691
##  [8]  0.04700 -0.00485 -0.11419
```

```r
# Compared to true values
c(true_betas_orig,true_betas_aug);
```

```
##  [1]  2.00 -2.00  1.00 -1.00 -1.00 -1.00  0.50  0.50 -0.25 -0.25
```

```r
# Posterior standard deviation
apply(beta_Standard,2,sd);
```

```
##  [1] 0.423 0.501 0.456 0.255 0.276 0.440 0.317 0.245 0.197 0.262
```

```r
apply(beta_NABAgnostic,2,sd);
```

```
##  [1] 0.230 0.250 0.215 0.207 0.140 0.369 0.202 0.124 0.109 0.144
```

```r
apply(beta_NABOptimist,2,sd);
```

```
##  [1] 0.2103 0.2288 0.1954 0.1928 0.1233 0.3588 0.1829 0.1124 0.0992 0.1369
```

```r
apply(beta_SABAgnostic,2,sd);
```

```
##  [1] 0.230 0.259 0.306 0.249 0.277 0.318 0.282 0.182 0.155 0.238
```

```r
apply(beta_SABOptimist,2,sd);
```

```
##  [1] 0.215 0.244 0.279 0.236 0.271 0.305 0.273 0.169 0.148 0.221
```

```r
# Root mean-squared error
matrix_true_beta = matrix(c(true_betas_orig,true_betas_aug),
                          nrow = mc_iter_after_warmup * mc_chains,
                          ncol = p + q,
                          byrow = T);
sqrt(mean(rowSums((beta_Standard - matrix_true_beta)^2)));
```

```
## [1] 1.87
```

```r
sqrt(mean(rowSums((beta_NABAgnostic - matrix_true_beta)^2)));
```

```
## [1] 1.73
```

```r
sqrt(mean(rowSums((beta_NABOptimist - matrix_true_beta)^2)));
```

```
## [1] 1.74
```

```r
sqrt(mean(rowSums((beta_SABAgnostic - matrix_true_beta)^2)));
```

```
## [1] 1.72
```

```r
sqrt(mean(rowSums((beta_SABOptimist - matrix_true_beta)^2)));
```

```
## [1] 1.73
```