#'Multinomial Probit Bayesian Additive Regression Trees

#'

#'Multinomial probit modeling using Bayesian Additive Regression Trees for Dynamic Data,

#'@param formula response ~ covariates,

#'@param data Training data with the multinomial response,

#'@param test.data Test Data with number of rows equals nsamp x ndraws, typically without the response. For the ith subject, (i, i+nsamp, ..., i+(ndraws-1)nsamp) rows are its simulated posterior outcomes from the previous simulation,

#'@param base order index of the reference level of the multinomial response. For example, if the response takes values 2, 3, and 4, then base = 2 sets response value 3 as the reference. Default is the highest class,

#'@param Prior List of Priors for MPBART: e.g., Prior = list(nu=p+2, V= diag(p - 1), ntrees=200, kfac=2.0, pswap=0, pbd=1.0, pb=0.5 , beta = 2.0, alpha = 0.95, nc = 100, minobsnode = 10).

#'The components of Prior are

#' \itemize{

#'\item nu : The covariance matrix of latent variables is assumed to have prior distribution Inv-Wish(nu, V). nu is the degree of freedom and nu > (nlatent - 1).

#'\item V : The positive definite scale matrix in the Inverse-Wishart prior of the covariance matrix.

#'\item ntrees : The total number of trees in each round of BART fitting.

#'\item kfac : A tuning parameter that satisfies mu - kfac \* sigma = ymin and mu + kfac \* sigma = ymax, where mu and sigma are the mean and std of the Gaussian prior distribution of the sum of fit of all trees.

#'\item pswap : The prior probability of swap move in simulating trees; default 0, there should be pswap no larger than 1-pbd.

#'\item pbd : The prior probability of birth/death move in simulating trees; default 1.

#'\item pb : The prior probability of birth given birth/death; default 0.5.

#'\item alpha : The prior probability of a bottom node splits is alpha/(1+d)^beta, d is depth of node.

#'\item beta : see alpha.

#'\item nc : The number of equally spaced cutpoints between min and max for each covariate.

#'\item minobsnode : The minimum number of observations in bottom nodes for birth in simulating trees.

#'}

#'@param Mcmc List of MCMC starting values, burn-in ...: e.g., list(sigma0 = diag(p - 1), burn = 100, ndraws = 1000, nSigDr = 50, keep\_sigma\_draws=FALSE)

#'The components of Mcmc are

#' \itemize{

#'\item sigma0 : The starting value of the covariance matrix of latent variables.

#'\item nSigDr: User-specified upper limit to repeated draws of the covariance variance matrix of latent variables in each round of posterior draw when condition 10 in Jiao and van Dyk 2015 is not satisfied. Default 50.

#'}

#'@return samp\_train ndraws x n posterior matrix of the training data outcome,

#'@return samp\_test ndraws x testn posterior matrix of the test data outcome,

#'@return sigmasample posterior samples of the latent variable covariance matrix.

#'@examples

#'##simulate data (example from Friedman MARS paper)

#'f = function(x){

#' 10\*sin(pi\*x[,1]\*x[,2]) + 20\*(x[,3]-.5)^2+10\*x[,4]+5\*x[,5]

#'}

#'sigma = 1.0 #y = f(x) + sigma\*z , z~N(0,1)

#'n = 100 #number of observations

#'set.seed(99)

#'###############################################

#'nd = 1000 # ndraws

#'x1 = matrix(runif(n\*9),n,9) #10 variables, only first 5 matter

#'x1 = x1[rep(1:n,nd),]

#'x2 = runif(n\*nd)

#'x = cbind(x1, x2)

#'Ey = f(x)

#'u=Ey+sigma\*rnorm(n\*nd)

#'z = (u-min(u))/(max(u)-min(u))

#'y = 1\*(z<0.4)+ 2\*(z>=0.4 & z<0.6) + 3\*(z>=0.6)

#'tdat = data.frame(x,y)

#'###############################################

#'x=matrix(runif(n\*10),n,10) #10 variables, only first 5 matter

#'Ey = f(x)

#'u=Ey+sigma\*rnorm(n)

#'z = (u-min(u))/(max(u)-min(u))

#'y = 1\*(z<0.4)+ 2\*(z>=0.4 & z<0.6) + 3\*(z>=0.6)

#'dat = data.frame(x,y)

#'colnames(tdat) = colnames(dat)

#'###############################################

#'p = 3

#'fml = as.formula("y ~ X1+X2+X3+X4+X5+X6+X7+X8+X9+X10")

#'bmpfit = DynMPBART\_call(fml, data = dat, test.data = tdat,

#' Prior = list(nu = p-1+3, V = diag(p-1),

#' ntrees = 100,

#' kfac = 2,

#' pswap = 0.1, pbd = 0.5, pb = 0.25,

#' alpha = 0.95, beta = 2.0,

#' nc = 100, minobsnode = 10),

#' Mcmc = list(sigma0 = diag(p-1), burn = 100, ndraws = nd,

#' nSigDr = 20, keep\_sigma\_draws=T))

#'

#'@import bayesm mlbench mlogit cvTools stats

#'@export

#'@useDynLib allBART

DynMPBART\_call <- function(formula, data, base = NULL,test.data = NULL,

Prior = NULL, Mcmc = NULL)

{

callT <- match.call(expand.dots = TRUE)

formula <- mFormula(formula)

response.name <- paste(deparse(attr(formula, "lhs")[[1L]]))

m <- match(c("formula", "data"), names(callT), 0L)

mf <- callT

mf <- mf[c(1L, m)]

mf$formula <- formula

mf[[1L]] <- as.name("model.frame")

mf <- eval(mf, parent.frame())

Y <- model.response(mf)

Y <- as.factor(Y)

lev <- levels(Y)

p <- length(lev)

if (p < 3){

stop("The number of alternatives should be at least 3.")

}

if (!is.null(base))

{

base <- base

} else {

base <- lev[p]

}

if (base <= p) {

Y <- relevel(Y, ref = base)

lev <- levels(Y)

} else {

stop(paste("Error: `base' does not exist in the response variable."))

}

base <- lev[1]

counts <- table(Y)

if (any(counts == 0)) {

warning(paste("group(s)", paste(lev[counts == 0], collapse = " "), "are empty"))

Y <- factor(Y, levels = lev[counts > 0])

lev <- lev[counts > 0]

}

Y <- as.numeric(unclass(Y)) - 1

Y <- ifelse(Y==0, p,Y)

cat("The base level is: '", lev[1], "'.\n\n", sep="")

relvelved <- c(lev[2:length(lev)], lev[1])

Terms <- attr(mf, "terms")

X <- model.matrix.default(Terms, mf)

xcolnames <- colnames(X)[-1]

if(length(xcolnames) == 1 ){

X <- data.frame(X[,xcolnames])

names(X) <- xcolnames[1]

} else {

X <- X[,xcolnames]

}

if (!is.null(test.data)){

if(is.null(Mcmc$ndraws))

stop(paste("Error: ndraws is required for dynamic model calls."))

ndraws=Mcmc$ndraws

testnsub = nrow(test.data) / ndraws

if(testnsub %% 1 != 0)

stop(paste("Error: testn does not equal to test.nsub x ndraws."))

if(length(xcolnames) == 1 ){

Xtest <- data.frame(test.data[,xcolnames])

names(Xtest) <- xcolnames[1]

} else {

Xtest <- test.data[,xcolnames]

}

} else {

stop(paste("Error: no test data, please use MPBART\_call instead."))

}

Data = list(p=p,y=Y,X= X)

testData = list(p=p,X= Xtest)

cat("Table of y values",fill=TRUE)

print(table(model.response(mf) ))

n=length(Data$y)

pm1=p-1

if (!is.null(test.data)){

testn <- nrow(testData$X)

} else {

testn <- 0

}

binaryX = rep(NA,ncol(Data$X))

for(i in 1:ncol(Data$X)){

binaryX[i] = 1\*(length(unique(Data$X[,i]))==2)

}

if(missing(Prior))

{nu=pm1+3; V=nu\*diag(pm1);

ntrees=200; kfac=2.0;pswap=0;pbd=1.0;pb=0.5;beta = 2.0;alpha = 0.95; nc = 100; minobsnode = 10;

}

else

{if(is.null(Prior$nu)) {nu=pm1+3} else {nu=Prior$nu}

if(is.null(Prior$V)) {V=nu\*diag(pm1)} else {V=Prior$V}

if(is.null(Prior$ntrees)) {ntrees=200} else {ntrees=Prior$ntrees}

if(is.null(Prior$kfac)) {kfac=2.0} else {kfac=Prior$kfac}

if(is.null(Prior$pswap)) {pswap=0.0} else {pswap=Prior$pswap}

if(is.null(Prior$pbd)) {pbd=1.0} else {pbd=Prior$pbd}

if(is.null(Prior$pb)) {pb=0.5} else {pb=Prior$pb}

if(is.null(Prior$beta)) {beta = 2.0} else {beta=Prior$beta}

if(is.null(Prior$alpha)) {alpha = 0.95} else {alpha=Prior$alpha}

if(is.null(Prior$nc)) {nc=100} else {nc=Prior$nc}

if(is.null(Prior$minobsnode)) {minobsnode= 10} else {minobsnode=Prior$minobsnode}

}

if(is.null(Mcmc$sigma0)) {sigma0=diag(pm1)} else {sigma0=Mcmc$sigma0}

if(is.null(Mcmc$burn)) {burn=100} else {burn=Mcmc$burn}

#if(is.null(Mcmc$ndraws)) {ndraws=1000} else {ndraws=Mcmc$ndraws}

if(is.null(Mcmc$nSigDr)) {nSigDr=50} else {nSigDr=Mcmc$nSigDr}

if(is.null(Mcmc$keep\_sigma\_draws)) {keep\_sigma\_draws=FALSE} else {keep\_sigma\_draws=Mcmc$keep\_sigma\_draws}

sigmai = solve(sigma0)

if( keep\_sigma\_draws==FALSE){

sigmasample = as.double(0);

savesigma = 0;

} else {

sigmasample = as.double(rep(sigma0, ndraws));

savesigma = 1;

}

cat("Number of trees: ", ntrees, ".\n\n", sep="")

cat("Number of draws: ", ndraws, ".\n\n", sep="")

cat("burn-in: ", burn, "'\n\n", sep="")

cat("testnsub: ", testnsub, "'\n\n", sep="")

res = .C('mydynmpbart',w=as.double(rep(0,nrow(Data$X))),

trainx= as.double(t(Data$X)),

testx= as.double(t(testData$X)),

mu = as.double(rep(0,nrow(Data$X))),

sigmai = as.double(sigmai),

V = as.double(V),

n = as.integer(length(Data$y)),

n\_dim = as.integer(ncol(sigmai)),

y = as.integer(Data$y),

n\_cov = as.integer(ncol(Data$X)),

nu = as.integer(nu),

testn = as.integer(testn),

testnsub = as.integer(testnsub),

ndraws = as.integer(ndraws),

burn = as.integer(burn),

ntrees = as.integer(ntrees),

nSigDr = as.integer(nSigDr),

kfac = as.double(kfac),

pswap = as.double(pswap),

pbd = as.double(pbd),

pb = as.double(pb),

alpha = as.double(alpha),

beta = as.double(beta),

nc = as.integer(nc),

savesigma = as.integer(savesigma),

minobsnode = as.integer(minobsnode),

sigmasample = sigmasample,

vec\_test = as.integer(rep(0,testnsub\*ndraws)),

vec\_train = as.integer(rep(0,n\*ndraws)),

binaryX = as.integer(binaryX))

relvelved = as.numeric(relvelved)

vec\_class\_train <- matrix(relvelved[res$vec\_train], nrow = n)

if (!is.null(test.data)){

vec\_class\_test <- matrix(relvelved[res$vec\_test], nrow = testnsub)

} else {

vec\_class\_test <- NULL

}

if(savesigma){

ret = list(samp\_test = vec\_class\_test,

samp\_train = vec\_class\_train,

sigmasample = res$sigmasample);

} else {

ret = list(samp\_test = vec\_class\_test,

samp\_train = vec\_class\_train);

}

return(ret)

}