cv.R

georges

2023-07-23

```
# getModelData <- function(model) {
#   # returns a data frame with the data to which the model was fit
#   # model: a statistical model object that responds to model.frame() and formula()
#   data1 <- data <- model.frame(model)
#   vars <- all.vars(formula(model))
#   if ("pi" %in% vars) {
#     vars <- setdiff(vars, "pi")
#     message("the symbol 'pi' is treated as a numeric constant in the model formula")
#   }
#   cols <- colnames(data)
#   check <- vars %in% cols
#   if (!(all(check))) {
#     missing.cols <- !check
#     data1 <- expand.model.frame(model, vars[missing.cols])
#   }
#   missing.cols <- !cols %in% colnames(data1)
#   if (any(missing.cols)) {
#     data1 <- cbind(data1, data[missing.cols])
#   }
#   cols <- colnames(data1)
#   valid <- make.names(cols) == cols | grepl("^\\(.*\\)$", cols)
#   data1[valid]
# }
```

Extract response variable(s)

Generic function to extract the response variable(s) from a fitted model

@param model a fitted model @param ... additional parameters for specific methods

@returns a vector or matrix containing the values of response variable(s)

@examples fit <- lm(cbind(hp, mpg) ~ gear, mtcars) getResponse(fit) @export

```
getResponse <- function(model, ...){
  UseMethod("getResponse")
}
```

@describeIn getResponse default method @export

```
getResponse.default <- function(model, ...){
  y <- model$y
  if (is.null(y)) y <- model.response(model.frame(model))
  if (!is.numeric(y)) stop("non-numeric response")
  y
}
```

```r
mse <- function(y, yhat){
  mean((y - yhat)^2)
}
```

@export

```r
BayesRule2 <- function(y, yhat){
  yhat <- round(yhat)
  mean(y != yhat) # proportion in error
}
```

@export

```r
BayesRule <- function(y, yhat){
  if (!all(y %in% c(0, 1))) stop("response values not all 0 or 1")
  if (any(yhat < 0) || any(yhat > 1)) stop("fitted values outside of interval [0, 1]")
  yhat <- round(yhat)
  mean(y != yhat) # proportion in error
}
```

@export

```r
cv <- function(model, data, criterion, k, seed, ...){
  # Cross Validation
  # Args:
  #   model: model object
  #   data: data frame to which the model was fit
  #   criterion: cross-validation criterion function
  #   k: perform k-fold cross-validation
  #   seed: for R's random number generator
  UseMethod("cv")
}
```

@export

```r
cv.default <- function(model, data=insight::get_data(model),
                       criterion=mse, k=nrow(data),
                       seed, parallel=FALSE,
                       ncores=parallelly::availableCores(logical=FALSE), ...){
  # Args:
  #   model: a model object that responds to model.frame(), update(), and predict()
  #.         and for which the response is stored in model$y or accessible via model.response()
  #   data: data frame to which the model was fit (not usually necessary)
  #   criterion: cross-validation criterion function of form f(y.obs, y.fitted)
  #              (default is mse)
  #   k: perform k-fold cross-validation (default is n-fold)
  #   seed: for R's random number generator
  #   parallel: do computations in parallel? (default is FALSE)
  #   ncores: number of cores to use for parallel computations
  #           (default is number of physical cores detected)
  #   ...: to match generic
  # Returns: a "cv" object with the cv criterion averaged across the folds,
  #          the bias-adjsuted averaged cv criterion,
  #          the criterion applied to the model fit to the full data set,
  #.         and the initial value of R's RNG seed
  f <- function(i){
```

```
    # helper function to compute cv criterion for each fold
    indices.i <- indices[starts[i]:ends[i]]
    model.i <- update(model, data=data[ - indices.i, ])
    fit.o.i <- predict(model.i, newdata=data, type="response")
    fit.i <- fit.o.i[indices.i]
    c(criterion(y[indices.i], fit.i), criterion(y, fit.o.i))
  }
  y <- getResponse(model)
  n <- nrow(data)
  if (!is.numeric(k) || length(k) > 1L || k > n || k < 2 || k != round(k)){
    stop("k must be an integer between 2 and n")
  }
  if (k != n){
    if (missing(seed)) seed <- sample(1e6, 1L)
    set.seed(seed)
    message("R RNG seed set to ", seed)
  } else {
    seed <- NULL
  }
  nk <-  n %/% k # number of cases in each fold
  rem <- n %% k  # remainder
  folds <- rep(nk, k) + c(rep(1, rem), rep(0, k - rem)) # allocate remainder
  ends <- cumsum(folds) # end of each fold
  starts <- c(1, ends + 1)[-(k + 1)] # start of each fold
  indices <- sample(n, n) # permute cases
  if (parallel && ncores > 1){
    if (!require("doParallel")) stop("doParallel package is missing")
    cl <- makeCluster(ncores)
    registerDoParallel(cl)
    result <- foreach(i = 1L:k, .combine=rbind) %dopar% {
      f(i)
    }
    stopCluster(cl)
  } else {
    result <- matrix(0, k, 2L)
    for (i in 1L:k){
      result[i, ] <- f(i)
    }
  }
  cv <- weighted.mean(result[, 1L], folds)
  cv.full <- criterion(y, fitted(model))
  adj.cv <- cv + cv.full - weighted.mean(result[, 2L], folds)
  result <- list("CV crit" = cv, "adj CV crit" = adj.cv, "full crit" = cv.full,
                 "k" = if (k == n) "n" else k, "seed" = seed)
  class(result) <- "cv"
  result
}
```

@export

```
print.cv <- function(x, ...){
  cat(x[["k"]], "-Fold Cross Validation", sep="")
  cat("\ncross-validation criterion =", x[["CV crit"]])
  cat("\nbias-adjusted cross-validation criterion =", x[["adj CV crit"]])
```

```
  cat("\nfull-sample criterion =", x[["full crit"]], "\n")
  invisible(x)
}
```

@export

```
cv.lm <- function(model, data=insight::get_data(model), criterion=mse, k=nrow(data),
                        seed, parallel=FALSE,
                        ncores=parallelly::availableCores(logical=FALSE), ...){
  UpdateLM <- function(omit){
    # compute coefficients with omit cases deleted
    #  uses the Woodbury matrix identity
    # <https://en.wikipedia.org/wiki/Woodbury_matrix_identity>
    x <- X[omit, , drop=FALSE]
    dg <- if (length(omit) > 1L) diag(1/w[omit]) else 1/w[omit]
    XXi.u <- XXi + (XXi %*% t(x) %*% solve(dg - x %*% XXi %*% t(x)) %*% x %*% XXi)
    b.u <- XXi.u %*% (Xy - t(X[omit, , drop=FALSE]) %*% (w[omit] * y[omit]))
    as.vector(b.u)
  }
  f <- function(i){
    # helper function to compute cv criterion for each fold
    indices.i <- indices[starts[i]:ends[i]]
    b.i <- UpdateLM(indices.i)
    fit.o.i <- X %*% b.i
    fit.i <- fit.o.i[indices.i]
    c(criterion(y[indices.i], fit.i), criterion(y, fit.o.i))
  }
  X <- model.matrix(model)
  y <- getResponse(model)
  w <- weights(model)
  if (is.null(w)) w <- rep(1, length(y))
  n <- nrow(data)
  b <- coef(model)
  p <- length(b)
  if (p > model$rank) {
    message(paste0("The model has ", if (sum(is.na(b)) == 1L) "an ",
                   "aliased coefficient", if (sum(is.na(b)) > 1L) "s", ":"))
    print(b[is.na(b)])
    message("Aliased coefficients removed from the model")
    X <- X[, !is.na(b)]
    p <- ncol(X)
    model <- lm.wfit(X, y, w)
  }
  XXi <- chol2inv(model$qr$qr[1L:p, 1L:p, drop = FALSE])
  Xy <- t(X) %*% (w * y)
  if (!is.numeric(k) || length(k) > 1L || k > n || k < 2 || k != round(k)){
    stop("k must be an integer between 2 and n")
  }
  if (k != n){
    if (missing(seed)) seed <- sample(1e6, 1L)
    set.seed(seed)
    message("R RNG seed set to ", seed)
  } else {
    seed <- NULL
```

```
  }
  nk <-  n %/% k # number of cases in each fold
  rem <- n %% k  # remainder
  folds <- rep(nk, k) + c(rep(1, rem), rep(0, k - rem)) # allocate remainder
  ends <- cumsum(folds) # end of each fold
  starts <- c(1, ends + 1)[-(k + 1)] # start of each fold
  indices <- sample(n, n) # permute cases
  if (parallel && ncores > 1L){
    if (!require("doParallel")) stop("doParallel package is missing")
    cl <- makeCluster(ncores)
    registerDoParallel(cl)
    result <- foreach(i = 1L:k, .combine=rbind) %dopar% {
      f(i)
    }
    stopCluster(cl)
  } else {
    result <- matrix(0, k, 2L)
    for (i in 1L:k){
      result[i, ] <- f(i)
    }
  }
  cv <- weighted.mean(result[, 1L], folds)
  cv.full <- criterion(y, fitted(model))
  adj.cv <- cv + cv.full - weighted.mean(result[, 2L], folds)
  result <- list("CV crit" = cv, "adj CV crit" = adj.cv, "full crit" = cv.full,
                 "k" = if (k == n) "n" else k, "seed" = seed)
  class(result) <- "cv"
  result
}
```

@export

```
cv.glm <- function(model, data=insight::get_data(model), criterion=mse, k=nrow(data),
                   seed, parallel=FALSE,
                   ncores=parallelly::availableCores(logical=FALSE),
                   approximate=FALSE, ...){
  UpdateIWLS <- function(omit){
    # compute coefficients with omit cases deleted
    #  uses the Woodbury matrix identity
    # <https://en.wikipedia.org/wiki/Woodbury_matrix_identity>
    x <- X[omit, , drop=FALSE]
    dg <- if (length(omit) > 1L) diag(1/w[omit]) else 1/w[omit]
    XXi.u <- XXi + (XXi %*% t(x) %*% solve(dg - x %*% XXi %*% t(x)) %*% x %*% XXi)
    b.u <- XXi.u %*% (Xz - t(X[omit, , drop=FALSE]) %*% (w[omit] * z[omit]))
    as.vector(b.u)
  }
  f <- function(i){
    # helper function to compute cv criterion for each fold
    indices.i <- indices[starts[i]:ends[i]]
    b.i <- UpdateIWLS(indices.i)
    fit.o.i <- linkinv(X %*% b.i)
    fit.i <- fit.o.i[indices.i]
    c(criterion(y[indices.i], fit.i), criterion(y, fit.o.i))
  }
```

```
n <- nrow(data)
if (k != n){
  if (missing(seed)) seed <- sample(1e6, 1L)
  set.seed(seed)
  if (approximate) message("R RNG seed set to ", seed)
} else {
  seed <- NULL
}
if (!approximate){
cv.default(model=model, data=data, criterion=criterion, k=k, seed=seed,
           parallel=parallel, ncores=ncores, ...)
} else {
  b <- coef(model)
  p <- length(b)
  w <- weights(model, type="working")
  X <- model.matrix(model)
  y <- getResponse(model)
  if (p > model$rank) {
    message(paste0("The model has ", if (sum(is.na(b)) == 1L) "an ",
                   "aliased coefficient", if (sum(is.na(b)) > 1L) "s", ":"))
    print(b[is.na(b)])
    message("Aliased coefficients removed from the model")
    X <- X[, !is.na(b)]
    p <- ncol(X)
  }
  eta <- predict(model)
  mu <-  fitted(model)
  z <- eta + (y - mu)/family(model)$mu.eta(eta)
  mod.lm <- lm.wfit(X, z, w)
  linkinv <- family(model)$linkinv
  XXi <- chol2inv(mod.lm$qr$qr[1L:p, 1L:p, drop = FALSE])
  Xz <- t(X) %*% (w * z)
  if (!is.numeric(k) || length(k) > 1L || k > n || k < 2 || k != round(k)){
    stop("k must be an integer between 2 and n")
  }
  nk <-  n %/% k # number of cases in each fold
  rem <- n %% k  # remainder
  folds <- rep(nk, k) + c(rep(1, rem), rep(0, k - rem)) # allocate remainder
  ends <- cumsum(folds) # end of each fold
  starts <- c(1, ends + 1)[-(k + 1)] # start of each fold
  indices <- sample(n, n) # permute cases
  if (parallel && ncores > 1L){
    if (!require("doParallel")) stop("doParallel package is missing")
    cl <- makeCluster(ncores)
    registerDoParallel(cl)
    result <- foreach(i = 1L:k, .combine=rbind) %dopar% {
      f(i)
    }
    stopCluster(cl)
  } else {
    result <- matrix(0, k, 2L)
    for (i in 1L:k){
      result[i, ] <- f(i)
```

```
      }
    }
    cv <- weighted.mean(result[, 1L], folds)
    cv.full <- criterion(y, fitted(model))
    adj.cv <- cv + cv.full - weighted.mean(result[, 2L], folds)
    result <- list("CV crit" = cv, "adj CV crit" = adj.cv, "full crit" = cv.full,
                   "k" = if (k == n) "n" else k, "seed" = seed)
    class(result) <- "cv"
    result
  }
}
```