KNN算法的元算法

function (train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)

{

train <- as.matrix(train)

if (is.null(dim(test)))

dim(test) <- c(1, length(test))

test <- as.matrix(test)

if (any(is.na(train)) || any(is.na(test)) || any(is.na(cl)))

stop("no missing values are allowed")

p <- ncol(train)

ntr <- nrow(train)

if (length(cl) != ntr)

stop("'train' and 'class' have different lengths")

if (ntr < k) {

warning(gettextf("k = %d exceeds number %d of patterns",

k, ntr), domain = NA)

k <- ntr

}

if (k < 1)

stop(gettextf("k = %d must be at least 1", k), domain = NA)

nte <- nrow(test)

if (ncol(test) != p)

stop("dims of 'test' and 'train' differ")

clf <- as.factor(cl)

nc <- max(unclass(clf))

Z <- .C(VR\_knn, as.integer(k), as.integer(l), as.integer(ntr),

as.integer(nte), as.integer(p), as.double(train), as.integer(unclass(clf)),

as.double(test), res = integer(nte), pr = double(nte),

integer(nc + 1), as.integer(nc), as.integer(FALSE), as.integer(use.all))

res <- factor(Z$res, levels = seq\_along(levels(clf)), labels = levels(clf))

if (prob)

attr(res, "prob") <- Z$pr

res

}

Kknn的元算法

function (formula = formula(train), train, test, na.action = na.omit(),

k = 7, distance = 2, kernel = "optimal", ykernel = NULL,

scale = TRUE, contrasts = c(unordered = "contr.dummy", ordered = "contr.ordinal"))

{

if (is.null(ykernel))

ykernel = 0

weight.y = function(l = 1, diff = 0) {

k = diff + 1

result = matrix(0, l, l)

diag(result) = k

for (i in 1:(k - 1)) {

for (j in 1:(l - i)) {

result[j, j + i] = k - i

result[j + i, j] = k - i

}

}

result

}

kernel <- match.arg(kernel, c("rectangular", "triangular",

"epanechnikov", "biweight", "triweight", "cos", "inv",

"gaussian", "rank", "optimal"), FALSE)

ca <- match.call()

response = NULL

old.contrasts <- getOption("contrasts")

options(contrasts = contrasts)

formula = as.formula(formula)

mf <- model.frame(formula, data = train)

mt <- attr(mf, "terms")

mt2 <- delete.response(mt)

cl <- model.response(mf)

d <- sum(attr(mt, "order"))

if (is.ordered(cl)) {

response <- "ordinal"

lev <- levels(cl)

}

if (is.numeric(cl))

response <- "continuous"

if (is.factor(cl) & !is.ordered(cl)) {

response <- "nominal"

lev <- levels(cl)

}

if (distance <= 0)

stop("distance must >0")

if (k <= 0)

stop("k must >0")

learn <- model.matrix(mt, mf)

valid <- model.matrix(mt2, test)

m <- dim(learn)[1]

p <- dim(valid)[1]

q <- dim(learn)[2]

ind <- attributes(learn)$assign

d.sd <- numeric(length(ind)) + 1

we <- numeric(length(ind)) + 1

d.sd = apply(learn, 2, stats::var)

for (i in unique(ind)) {

d.sd[ind == i] = sqrt(mean(d.sd[ind == i]))

we[ind == i] = 1/sum(ind == i)

}

we[d.sd == 0] = 0

d.sd[d.sd == 0] = 1

if (scale) {

learn <- sweep(learn, 2L, d.sd, "/", check.margin = FALSE)

valid <- sweep(valid, 2L, d.sd, "/", check.margin = FALSE)

}

ord = order(we \* apply(learn, 2, sd), decreasing = TRUE)

we = we[ord]

learn = learn[, ord, drop = FALSE]

valid = valid[, ord, drop = FALSE]

Euclid <- FALSE

if (distance == 2)

Euclid <- TRUE

if (Euclid)

dmtmp <- .C("dmEuclid", as.double(learn), as.double(valid),

as.integer(m), as.integer(p), as.integer(q), dm = double((k +

1L) \* p), cl = integer((k + 1L) \* p), k = as.integer(k +

1), as.double(distance), as.double(we), PACKAGE = "kknn")

else dmtmp <- .C("dm", as.double(learn), as.double(valid),

as.integer(m), as.integer(p), as.integer(q), dm = double((k +

1L) \* p), cl = integer((k + 1L) \* p), k = as.integer(k +

1), as.double(distance), as.double(we), PACKAGE = "kknn")

D <- matrix(dmtmp$dm, nrow = p, ncol = k + 1)

C <- matrix(dmtmp$cl, nrow = p, ncol = k + 1)

maxdist <- D[, k + 1]

maxdist[maxdist < 1e-06] <- 1e-06

D <- D[, 1:k]

C <- C[, 1:k] + 1

CL <- matrix(cl[C], nrow = p, ncol = k)

if (response != "continuous") {

l <- length(lev)

weightClass <- matrix(0, p, l)

}

if (response == "continuous") {

weightClass <- NULL

}

W <- D/maxdist

W <- pmin(W, 1 - (1e-06))

W <- pmax(W, 1e-06)

if (kernel == "rank")

W <- (k + 1) - t(apply(as.matrix(D), 1, rank))

if (kernel == "inv")

W <- 1/W

if (kernel == "rectangular")

W <- matrix(1, nrow = p, ncol = k)

if (kernel == "triangular")

W <- 1 - W

if (kernel == "epanechnikov")

W <- 0.75 \* (1 - W^2)

if (kernel == "biweight")

W <- dbeta((W + 1)/2, 3, 3)

if (kernel == "triweight")

W <- dbeta((W + 1)/2, 4, 4)

if (kernel == "cos")

W <- cos(W \* pi/2)

if (kernel == "triweights")

W <- 1

if (kernel == "gaussian") {

alpha = 1/(2 \* (k + 1))

qua = abs(qnorm(alpha))

W = W \* qua

W = dnorm(W, sd = 1)

}

if (kernel == "optimal") {

W = rep(optKernel(k, d = d), each = p)

}

W <- matrix(W, p, k)

if (response != "continuous") {

for (i in 1:l) {

weightClass[, i] <- rowSums(W \* (CL == lev[i]))

}

weightClass <- weightClass/rowSums(weightClass)

colnames(weightClass) <- lev

}

if (response == "ordinal") {

blub = length(lev)

weightClass = weightClass %\*% weight.y(blub, ykernel)

weightClass <- weightClass/rowSums(weightClass)

weightClass <- t(apply(weightClass, 1, cumsum))

colnames(weightClass) <- lev

fit <- numeric(p)

for (i in 1:p) fit[i] <- min((1:l)[weightClass[i, ] >=

0.5])

fit <- ordered(fit, levels = 1:l, labels = lev)

}

if (response == "nominal") {

fit <- apply(weightClass, 1, order, decreasing = TRUE)[1,

]

fit <- factor(fit, levels = 1:l, labels = lev)

if (kernel == "rectangular" && k > 1) {

blub <- apply(weightClass, 1, rank, ties.method = "max")

indices = (1:p)[colSums(blub == l) > 1]

blub = t(blub)

nM = matrix(0, p, l)

colnames(nM) = lev

for (i in 1:l) nM[, i] = apply((CL == lev[i]) %\*%

diag(1:k), 1, max)

nM = (blub == l) \* nM

nM[nM == 0] <- k + 1

fitv = numeric(p)

for (i in indices) fitv[i] = which(nM[i, ] == min(nM[i,

]))

fit[indices] <- factor(fitv[indices], levels = 1:l,

labels = lev)

}

}

if (response == "continuous")

fit <- rowSums(W \* CL)/pmax(rowSums(W), 1e-06)

options(contrasts = old.contrasts)

result <- list(fitted.values = fit, CL = CL, W = W, D = D,

C = C, prob = weightClass, response = response, distance = distance,

call = ca, terms = mt)

class(result) = "kknn"

result

}

Rpart:

function (formula, data, weights, subset, na.action = na.rpart,

method, model = FALSE, x = FALSE, y = TRUE, parms, control,

cost, ...)

{

Call <- match.call()

if (is.data.frame(model)) {

m <- model

model <- FALSE

}

else {

indx <- match(c("formula", "data", "weights", "subset"),

names(Call), nomatch = 0L)

if (indx[1] == 0L)

stop("a 'formula' argument is required")

temp <- Call[c(1L, indx)]

temp$na.action <- na.action

temp[[1L]] <- quote(stats::model.frame)

m <- eval.parent(temp)

}

Terms <- attr(m, "terms")

if (any(attr(Terms, "order") > 1L))

stop("Trees cannot handle interaction terms")

Y <- model.response(m)

wt <- model.weights(m)

if (any(wt < 0))

stop("negative weights not allowed")

if (!length(wt))

wt <- rep(1, nrow(m))

offset <- model.offset(m)

X <- rpart.matrix(m)

nobs <- nrow(X)

nvar <- ncol(X)

if (missing(method)) {

method <- if (is.factor(Y) || is.character(Y))

"class"

else if (inherits(Y, "Surv"))

"exp"

else if (is.matrix(Y))

"poisson"

else "anova"

}

if (is.list(method)) {

mlist <- method

method <- "user"

init <- if (missing(parms))

mlist$init(Y, offset, wt = wt)

else mlist$init(Y, offset, parms, wt)

keep <- rpartcallback(mlist, nobs, init)

method.int <- 4L

parms <- init$parms

}

else {

method.int <- pmatch(method, c("anova", "poisson", "class",

"exp"))

if (is.na(method.int))

stop("Invalid method")

method <- c("anova", "poisson", "class", "exp")[method.int]

if (method.int == 4L)

method.int <- 2L

init <- if (missing(parms))

get(paste("rpart", method, sep = "."), envir = environment())(Y,

offset, , wt)

else get(paste("rpart", method, sep = "."), envir = environment())(Y,

offset, parms, wt)

ns <- asNamespace("rpart")

if (!is.null(init$print))

environment(init$print) <- ns

if (!is.null(init$summary))

environment(init$summary) <- ns

if (!is.null(init$text))

environment(init$text) <- ns

}

Y <- init$y

xlevels <- .getXlevels(Terms, m)

cats <- rep(0L, ncol(X))

if (!is.null(xlevels))

cats[match(names(xlevels), colnames(X))] <- unlist(lapply(xlevels,

length))

extraArgs <- list(...)

if (length(extraArgs)) {

controlargs <- names(formals(rpart.control))

indx <- match(names(extraArgs), controlargs, nomatch = 0L)

if (any(indx == 0L))

stop(gettextf("Argument %s not matched", names(extraArgs)[indx ==

0L]), domain = NA)

}

controls <- rpart.control(...)

if (!missing(control))

controls[names(control)] <- control

xval <- controls$xval

if (is.null(xval) || (length(xval) == 1L && xval == 0L) ||

method == "user") {

xgroups <- 0L

xval <- 0L

}

else if (length(xval) == 1L) {

xgroups <- sample(rep(1L:xval, length = nobs), nobs,

replace = FALSE)

}

else if (length(xval) == nobs) {

xgroups <- xval

xval <- length(unique(xgroups))

}

else {

if (!is.null(attr(m, "na.action"))) {

temp <- as.integer(attr(m, "na.action"))

xval <- xval[-temp]

if (length(xval) == nobs) {

xgroups <- xval

xval <- length(unique(xgroups))

}

else stop("Wrong length for 'xval'")

}

else stop("Wrong length for 'xval'")

}

if (missing(cost))

cost <- rep(1, nvar)

else {

if (length(cost) != nvar)

stop("Cost vector is the wrong length")

if (any(cost <= 0))

stop("Cost vector must be positive")

}

tfun <- function(x) if (is.matrix(x))

rep(is.ordered(x), ncol(x))

else is.ordered(x)

labs <- sub("^`(.\*)`$", "\\1", attr(Terms, "term.labels"))

isord <- unlist(lapply(m[labs], tfun))

storage.mode(X) <- "double"

storage.mode(wt) <- "double"

temp <- as.double(unlist(init$parms))

if (!length(temp))

temp <- 0

rpfit <- .Call(C\_rpart, ncat = as.integer(cats \* (!isord)),

method = as.integer(method.int), as.double(unlist(controls)),

temp, as.integer(xval), as.integer(xgroups), as.double(t(init$y)),

X, wt, as.integer(init$numy), as.double(cost))

nsplit <- nrow(rpfit$isplit)

ncat <- if (!is.null(rpfit$csplit))

nrow(rpfit$csplit)

else 0L

if (nsplit == 0L)

xval <- 0L

numcp <- ncol(rpfit$cptable)

temp <- if (nrow(rpfit$cptable) == 3L)

c("CP", "nsplit", "rel error")

else c("CP", "nsplit", "rel error", "xerror", "xstd")

dimnames(rpfit$cptable) <- list(temp, 1L:numcp)

tname <- c("<leaf>", colnames(X))

splits <- matrix(c(rpfit$isplit[, 2:3], rpfit$dsplit), ncol = 5L,

dimnames = list(tname[rpfit$isplit[, 1L] + 1L], c("count",

"ncat", "improve", "index", "adj")))

index <- rpfit$inode[, 2L]

nadd <- sum(isord[rpfit$isplit[, 1L]])

if (nadd > 0L) {

newc <- matrix(0L, nadd, max(cats))

cvar <- rpfit$isplit[, 1L]

indx <- isord[cvar]

cdir <- splits[indx, 2L]

ccut <- floor(splits[indx, 4L])

splits[indx, 2L] <- cats[cvar[indx]]

splits[indx, 4L] <- ncat + 1L:nadd

for (i in 1L:nadd) {

newc[i, 1L:(cats[(cvar[indx])[i]])] <- -as.integer(cdir[i])

newc[i, 1L:ccut[i]] <- as.integer(cdir[i])

}

catmat <- if (ncat == 0L)

newc

else {

cs <- rpfit$csplit

ncs <- ncol(cs)

ncc <- ncol(newc)

if (ncs < ncc)

cs <- cbind(cs, matrix(0L, nrow(cs), ncc - ncs))

rbind(cs, newc)

}

ncat <- ncat + nadd

}

else catmat <- rpfit$csplit

if (nsplit == 0L) {

frame <- data.frame(row.names = 1L, var = "<leaf>", n = rpfit$inode[,

5L], wt = rpfit$dnode[, 3L], dev = rpfit$dnode[,

1L], yval = rpfit$dnode[, 4L], complexity = rpfit$dnode[,

2L], ncompete = 0L, nsurrogate = 0L)

}

else {

temp <- ifelse(index == 0L, 1L, index)

svar <- ifelse(index == 0L, 0L, rpfit$isplit[temp, 1L])

frame <- data.frame(row.names = rpfit$inode[, 1L], var = tname[svar +

1L], n = rpfit$inode[, 5L], wt = rpfit$dnode[, 3L],

dev = rpfit$dnode[, 1L], yval = rpfit$dnode[, 4L],

complexity = rpfit$dnode[, 2L], ncompete = pmax(0L,

rpfit$inode[, 3L] - 1L), nsurrogate = rpfit$inode[,

4L])

}

if (method.int == 3L) {

numclass <- init$numresp - 2L

nodeprob <- rpfit$dnode[, numclass + 5L]/sum(wt)

temp <- pmax(1L, init$counts)

temp <- rpfit$dnode[, 4L + (1L:numclass)] %\*% diag(init$parms$prior/temp)

yprob <- temp/rowSums(temp)

yval2 <- matrix(rpfit$dnode[, 4L + (0L:numclass)], ncol = numclass +

1L)

frame$yval2 <- cbind(yval2, yprob, nodeprob)

}

else if (init$numresp > 1L)

frame$yval2 <- rpfit$dnode[, -(1L:3L), drop = FALSE]

if (is.null(init$summary))

stop("Initialization routine is missing the 'summary' function")

functions <- if (is.null(init$print))

list(summary = init$summary)

else list(summary = init$summary, print = init$print)

if (!is.null(init$text))

functions <- c(functions, list(text = init$text))

if (method == "user")

functions <- c(functions, mlist)

where <- rpfit$which

names(where) <- row.names(m)

ans <- list(frame = frame, where = where, call = Call, terms = Terms,

cptable = t(rpfit$cptable), method = method, parms = init$parms,

control = controls, functions = functions, numresp = init$numresp)

if (nsplit)

ans$splits = splits

if (ncat > 0L)

ans$csplit <- catmat + 2L

if (nsplit)

ans$variable.importance <- importance(ans)

if (model) {

ans$model <- m

if (missing(y))

y <- FALSE

}

if (y)

ans$y <- Y

if (x) {

ans$x <- X

ans$wt <- wt

}

ans$ordered <- isord

if (!is.null(attr(m, "na.action")))

ans$na.action <- attr(m, "na.action")

if (!is.null(xlevels))

attr(ans, "xlevels") <- xlevels

if (method == "class")

attr(ans, "ylevels") <- init$ylevels

class(ans) <- "rpart"

ans

}

Lm:

function (formula, data, subset, weights, na.action, method = "qr",

model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,

contrasts = NULL, offset, ...)

{

ret.x <- x

ret.y <- y

cl <- match.call()

mf <- match.call(expand.dots = FALSE)

m <- match(c("formula", "data", "subset", "weights", "na.action",

"offset"), names(mf), 0L)

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

mf$drop.unused.levels <- TRUE

mf[[1L]] <- quote(stats::model.frame)

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

if (method == "model.frame")

return(mf)

else if (method != "qr")

warning(gettextf("method = '%s' is not supported. Using 'qr'",

method), domain = NA)

mt <- attr(mf, "terms")

y <- model.response(mf, "numeric")

w <- as.vector(model.weights(mf))

if (!is.null(w) && !is.numeric(w))

stop("'weights' must be a numeric vector")

offset <- as.vector(model.offset(mf))

if (!is.null(offset)) {

if (length(offset) != NROW(y))

stop(gettextf("number of offsets is %d, should equal %d (number of observations)",

length(offset), NROW(y)), domain = NA)

}

if (is.empty.model(mt)) {

x <- NULL

z <- list(coefficients = if (is.matrix(y)) matrix(, 0,

3) else numeric(), residuals = y, fitted.values = 0 \*

y, weights = w, rank = 0L, df.residual = if (!is.null(w)) sum(w !=

0) else if (is.matrix(y)) nrow(y) else length(y))

if (!is.null(offset)) {

z$fitted.values <- offset

z$residuals <- y - offset

}

}

else {

x <- model.matrix(mt, mf, contrasts)

z <- if (is.null(w))

lm.fit(x, y, offset = offset, singular.ok = singular.ok,

...)

else lm.wfit(x, y, w, offset = offset, singular.ok = singular.ok,

...)

}

class(z) <- c(if (is.matrix(y)) "mlm", "lm")

z$na.action <- attr(mf, "na.action")

z$offset <- offset

z$contrasts <- attr(x, "contrasts")

z$xlevels <- .getXlevels(mt, mf)

z$call <- cl

z$terms <- mt

if (model)

z$model <- mf

if (ret.x)

z$x <- x

if (ret.y)

z$y <- y

if (!qr)

z$qr <- NULL

z

}

Glm:

function (formula, family = gaussian, data, weights, subset,

na.action, start = NULL, etastart, mustart, offset, control = list(...),

model = TRUE, method = "glm.fit", x = FALSE, y = TRUE, contrasts = NULL,

...)

{

call <- match.call()

if (is.character(family))

family <- get(family, mode = "function", envir = parent.frame())

if (is.function(family))

family <- family()

if (is.null(family$family)) {

print(family)

stop("'family' not recognized")

}

if (missing(data))

data <- environment(formula)

mf <- match.call(expand.dots = FALSE)

m <- match(c("formula", "data", "subset", "weights", "na.action",

"etastart", "mustart", "offset"), names(mf), 0L)

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

mf$drop.unused.levels <- TRUE

mf[[1L]] <- quote(stats::model.frame)

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

if (identical(method, "model.frame"))

return(mf)

if (!is.character(method) && !is.function(method))

stop("invalid 'method' argument")

if (identical(method, "glm.fit"))

control <- do.call("glm.control", control)

mt <- attr(mf, "terms")

Y <- model.response(mf, "any")

if (length(dim(Y)) == 1L) {

nm <- rownames(Y)

dim(Y) <- NULL

if (!is.null(nm))

names(Y) <- nm

}

X <- if (!is.empty.model(mt))

model.matrix(mt, mf, contrasts)

else matrix(, NROW(Y), 0L)

weights <- as.vector(model.weights(mf))

if (!is.null(weights) && !is.numeric(weights))

stop("'weights' must be a numeric vector")

if (!is.null(weights) && any(weights < 0))

stop("negative weights not allowed")

offset <- as.vector(model.offset(mf))

if (!is.null(offset)) {

if (length(offset) != NROW(Y))

stop(gettextf("number of offsets is %d should equal %d (number of observations)",

length(offset), NROW(Y)), domain = NA)

}

mustart <- model.extract(mf, "mustart")

etastart <- model.extract(mf, "etastart")

fit <- eval(call(if (is.function(method)) "method" else method,

x = X, y = Y, weights = weights, start = start, etastart = etastart,

mustart = mustart, offset = offset, family = family,

control = control, intercept = attr(mt, "intercept") >

0L))

if (length(offset) && attr(mt, "intercept") > 0L) {

fit2 <- eval(call(if (is.function(method)) "method" else method,

x = X[, "(Intercept)", drop = FALSE], y = Y, weights = weights,

offset = offset, family = family, control = control,

intercept = TRUE))

if (!fit2$converged)

warning("fitting to calculate the null deviance did not converge -- increase 'maxit'?")

fit$null.deviance <- fit2$deviance

}

if (model)

fit$model <- mf

fit$na.action <- attr(mf, "na.action")

if (x)

fit$x <- X

if (!y)

fit$y <- NULL

fit <- c(fit, list(call = call, formula = formula, terms = mt,

data = data, offset = offset, control = control, method = method,

contrasts = attr(X, "contrasts"), xlevels = .getXlevels(mt,

mf)))

class(fit) <- c(fit$class, c("glm", "lm"))

fit

}