Package 'gwselect'

April 17, 2013

Type Package

Title Variable selection in Geographically Weighted Regression models

Version 0.1

Date 2012-08-24

Author Wesley Brooks

Maintainer Wesley Brooks <wbrooks2@wisc.edu>

Description Several variable selection techniques for GWR models

License GPL-3

Depends R (>= 2.14.0), sp, plotrix, glmnet, lars, ggplot2, doMC,scales

R topics documented:

`gwselect-package` *Variable selection for a geographically weighted regression model, using the LASSO*

Description

Variable selection for a geographically weighted regression model, using the LASSO

Details

Package:	gwselect
Type:	Package
Version:	0.1
Date:	2012-08-24
License:	GPL-3

~~ An overview of how to use the package, including the most important functions ~~

Author(s)

Wesley Brooks Maintainer: Wesley Brooks <wesley@somesquares.com>

References

~~ Literature or other references for background information ~~

See Also

~~ Optional links to other man pages, e.g. ~~ ~~ <pkg> ~~

`gwglmnet` *Fit a GW-GLM model using the LASSO for variable selection.*

Description

Fit a GW-GLM model using the LASSO for variable selection.

Usage

```
gwglmnet(formula, data, coords, gweight, bw, D = NULL, verbose = FALSE, longlat = FALSE, adapt =
```

Arguments

`formula`

`data`

`coords`

`gweight`

bw

D

verbose

longlat

adapt

s

family

weights

nearest.neighbors

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data, coords, gweight, bw, D = NULL, verbose = FALSE,
    longlat = FALSE, adapt = FALSE, s, family, weights = NULL,
    nearest.neighbors = FALSE)
{
    if (!is.logical(adapt))
        stop("adapt must be logical")
    if (is(data, "Spatial")) {
        if (!missing(coords))
            warning("data is Spatial* object, ignoring coords argument")
        coords <- coordinates(data)
        if ((is.null(longlat) || !is.logical(longlat)) && !is.na(is.projected(data)) &&
            !is.projected(data)) {
            longlat <- TRUE
        }
        else longlat <- FALSE
        data <- as(data, "data.frame")
    }
    if (is.null(longlat) || !is.logical(longlat))
        longlat <- FALSE
    if (missing(coords))
        stop("Observation coordinates have to be given")
    mf <- match.call(expand.dots = FALSE)
    m <- match(c("formula", "data"), names(mf), 0)
    mf <- mf[c(1, m)]
    mf$drop.unused.levels <- TRUE
    mf[[1]] <- as.name("model.frame")
    mf <- eval(mf, parent.frame())
    mt <- attr(mf, "terms")
    dp.n <- length(model.extract(mf, "response"))
    if (!is.null(weights) && !is.numeric(weights))
        stop("'weights' must be a numeric vector")
    if (is.null(weights))
```

```
    weights <- rep(as.numeric(1), dp.n)
if (any(is.na(weights)))
    stop("NAs in weights")
if (any(weights < 0))
    stop("negative weights")
y <- model.extract(mf, "response")
x <- model.matrix(mt, mf)
if (is.null(D)) {
    n = dim(coords)[1]
    if (longlat) {
        D = as.matrix(earth.dist(coords), n, n)
    }
    else {
        Xmat = matrix(rep(coords[, 1], times = n), n, n)
        Ymat = matrix(rep(coords[, 2], times = n), n, n)
        D = sqrt((Xmat - t(Xmat))^2 + (Ymat - t(Ymat))^2)
    }
}
if (!nearest.neighbors) {
    weight.matrix = gweight(D, bw)
}
else {
    n = dim(D)[1]
    bandwidths = sapply(1:n, function(x) {
        neighbor.weight(q = bw, D = D[x, ], weight.function = gweight,
            verbose = verbose, tol = 0.001)
    })
    weight.matrix = as.matrix(rbind(sapply(1:n, function(k) {
        gweight(as.vector(D[k, ]), as.numeric(bandwidths[1,
            k]))
    })), n, n)
}
if (!adapt) {
    res = gwglmnet.fit(x, y, coords, weight.matrix, s, verbose,
        family, weights)
}
else {
    res = gwglmnet.adaptive.fit(x, y, coords, weight.matrix,
        s, verbose, family, weights)
}
res[["data"]] = data
res[["response"]] = as.character(formula[[2]])
res
}
```

---

gwglmnet *Fit a GW-GLM model using the LASSO for variable selection.*

Description

Fit a GW-GLM model using the LASSO for variable selection.

Usage

```
gwglmnet(formula, data, coords, gweight, bw, D = NULL, verbose = FALSE, longlat = FALSE, adapt =
```

Arguments

formula

data

coords

gweight

bw

D

verbose

longlat

adapt

s

family

weights

nearest.neighbors

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data, coords, gweight, bw, D = NULL, verbose = FALSE,
    longlat = FALSE, adapt = FALSE, s, family, weights = NULL,
    nearest.neighbors = FALSE)
{
    if (!is.logical(adapt))
        stop("adapt must be logical")
    if (is(data, "Spatial")) {
        if (!missing(coords))
            warning("data is Spatial* object, ignoring coords argument")
        coords <- coordinates(data)
        if ((is.null(longlat) || !is.logical(longlat)) && !is.na(is.projected(data)) &&
            !is.projected(data)) {
            longlat <- TRUE
        }
        else longlat <- FALSE
        data <- as(data, "data.frame")
    }
    if (is.null(longlat) || !is.logical(longlat))
        longlat <- FALSE
    if (missing(coords))
        stop("Observation coordinates have to be given")
    mf <- match.call(expand.dots = FALSE)
    m <- match(c("formula", "data"), names(mf), 0)
    mf <- mf[c(1, m)]
```

```
    mf$drop.unused.levels <- TRUE
    mf[[1]] <- as.name("model.frame")
    mf <- eval(mf, parent.frame())
    mt <- attr(mf, "terms")
    dp.n <- length(model.extract(mf, "response"))
    if (!is.null(weights) && !is.numeric(weights))
        stop("'weights' must be a numeric vector")
    if (is.null(weights))
        weights <- rep(as.numeric(1), dp.n)
    if (any(is.na(weights)))
        stop("NAs in weights")
    if (any(weights < 0))
        stop("negative weights")
    y <- model.extract(mf, "response")
    x <- model.matrix(mt, mf)
    if (is.null(D)) {
        n = dim(coords)[1]
        if (longlat) {
            D = as.matrix(earth.dist(coords), n, n)
        }
        else {
            Xmat = matrix(rep(coords[, 1], times = n), n, n)
            Ymat = matrix(rep(coords[, 2], times = n), n, n)
            D = sqrt((Xmat - t(Xmat))^2 + (Ymat - t(Ymat))^2)
        }
    }
    if (!nearest.neighbors) {
        weight.matrix = gweight(D, bw)
    }
    else {
        n = dim(D)[1]
        bandwidths = sapply(1:n, function(x) {
            neighbor.weight(q = bw, D = D[x, ], weight.function = gweight,
                verbose = verbose, tol = 0.001)
        })
        weight.matrix = as.matrix(rbind(sapply(1:n, function(k) {
            gweight(as.vector(D[k, ]), as.numeric(bandwidths[1,
                k]))
        })), n, n)
    }
    if (!adapt) {
        res = gwglmnet.fit(x, y, coords, weight.matrix, s, verbose,
            family, weights)
    }
    else {
        res = gwglmnet.adaptive.fit(x, y, coords, weight.matrix,
            s, verbose, family, weights)
    }
    res[["data"]] = data
    res[["response"]] = as.character(formula[[2]])
    res
}
```

---

gwglmnet *Fit a GW-GLM model using the LASSO for variable selection.*

Description

Fit a GW-GLM model using the LASSO for variable selection.

Usage

```
gwglmnet(formula, data, coords, gweight, bw, D = NULL, verbose = FALSE, longlat = FALSE, adapt =
```

Arguments

formula

data

coords

gweight

bw

D

verbose

longlat

adapt

s

family

weights

nearest.neighbors

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data, coords, gweight, bw, D = NULL, verbose = FALSE,
    longlat = FALSE, adapt = FALSE, s, family, weights = NULL,
    nearest.neighbors = FALSE)
{
    if (!is.logical(adapt))
        stop("adapt must be logical")
    if (is(data, "Spatial")) {
        if (!missing(coords))
            warning("data is Spatial* object, ignoring coords argument")
        coords <- coordinates(data)
        if ((is.null(longlat) || !is.logical(longlat)) && !is.na(is.projected(data)) &&
            !is.projected(data)) {
            longlat <- TRUE
        }
        else longlat <- FALSE
        data <- as(data, "data.frame")
```

```
}
if (is.null(longlat) || !is.logical(longlat))
    longlat <- FALSE
if (missing(coords))
    stop("Observation coordinates have to be given")
mf <- match.call(expand.dots = FALSE)
m <- match(c("formula", "data"), names(mf), 0)
mf <- mf[c(1, m)]
mf$drop.unused.levels <- TRUE
mf[[1]] <- as.name("model.frame")
mf <- eval(mf, parent.frame())
mt <- attr(mf, "terms")
dp.n <- length(model.extract(mf, "response"))
if (!is.null(weights) && !is.numeric(weights))
    stop("'weights' must be a numeric vector")
if (is.null(weights))
    weights <- rep(as.numeric(1), dp.n)
if (any(is.na(weights)))
    stop("NAs in weights")
if (any(weights < 0))
    stop("negative weights")
y <- model.extract(mf, "response")
x <- model.matrix(mt, mf)
if (is.null(D)) {
    n = dim(coords)[1]
    if (longlat) {
        D = as.matrix(earth.dist(coords), n, n)
    }
    else {
        Xmat = matrix(rep(coords[, 1], times = n), n, n)
        Ymat = matrix(rep(coords[, 2], times = n), n, n)
        D = sqrt((Xmat - t(Xmat))^2 + (Ymat - t(Ymat))^2)
    }
}
if (!nearest.neighbors) {
    weight.matrix = gweight(D, bw)
}
else {
    n = dim(D)[1]
    bandwidths = sapply(1:n, function(x) {
        neighbor.weight(q = bw, D = D[x, ], weight.function = gweight,
            verbose = verbose, tol = 0.001)
    })
    weight.matrix = as.matrix(rbind(sapply(1:n, function(k) {
        gweight(as.vector(D[k, ]), as.numeric(bandwidths[1,
            k]))
    })), n, n)
}
if (!adapt) {
    res = gwglmnet.fit(x, y, coords, weight.matrix, s, verbose,
        family, weights)
}
else {
    res = gwglmnet.adaptive.fit(x, y, coords, weight.matrix,
        s, verbose, family, weights)
}
res[["data"]] = data
```

```
    res[["response"]] = as.character(formula[[2]])
    res
  }
```

---

gwglmnet *Fit a GW-GLM model using the LASSO for variable selection.*

Description

Fit a GW-GLM model using the LASSO for variable selection.

Usage

```
gwglmnet(formula, data, coords, gweight, bw, D = NULL, verbose = FALSE, longlat = FALSE, adapt =
```

Arguments

formula

data

coords

gweight

bw

D

verbose

longlat

adapt

s

family

weights

nearest.neighbors

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data, coords, gweight, bw, D = NULL, verbose = FALSE,
    longlat = FALSE, adapt = FALSE, s, family, weights = NULL,
    nearest.neighbors = FALSE)
{
    if (!is.logical(adapt))
        stop("adapt must be logical")
```

```
if (is(data, "Spatial")) {
    if (!missing(coords))
        warning("data is Spatial* object, ignoring coords argument")
    coords <- coordinates(data)
    if ((is.null(longlat) || !is.logical(longlat)) && !is.na(is.projected(data)) &&
        !is.projected(data)) {
        longlat <- TRUE
    }
    else longlat <- FALSE
    data <- as(data, "data.frame")
}
if (is.null(longlat) || !is.logical(longlat))
    longlat <- FALSE
if (missing(coords))
    stop("Observation coordinates have to be given")
mf <- match.call(expand.dots = FALSE)
m <- match(c("formula", "data"), names(mf), 0)
mf <- mf[c(1, m)]
mf$drop.unused.levels <- TRUE
mf[[1]] <- as.name("model.frame")
mf <- eval(mf, parent.frame())
mt <- attr(mf, "terms")
dp.n <- length(model.extract(mf, "response"))
if (!is.null(weights) && !is.numeric(weights))
    stop("'weights' must be a numeric vector")
if (is.null(weights))
    weights <- rep(as.numeric(1), dp.n)
if (any(is.na(weights)))
    stop("NAs in weights")
if (any(weights < 0))
    stop("negative weights")
y <- model.extract(mf, "response")
x <- model.matrix(mt, mf)
if (is.null(D)) {
    n = dim(coords)[1]
    if (longlat) {
        D = as.matrix(earth.dist(coords), n, n)
    }
    else {
        Xmat = matrix(rep(coords[, 1], times = n), n, n)
        Ymat = matrix(rep(coords[, 2], times = n), n, n)
        D = sqrt((Xmat - t(Xmat))^2 + (Ymat - t(Ymat))^2)
    }
}
if (!nearest.neighbors) {
    weight.matrix = gweight(D, bw)
}
else {
    n = dim(D)[1]
    bandwidths = sapply(1:n, function(x) {
        neighbor.weight(q = bw, D = D[x, ], weight.function = gweight,
            verbose = verbose, tol = 0.001)
    })
    weight.matrix = as.matrix(rbind(sapply(1:n, function(k) {
        gweight(as.vector(D[k, ]), as.numeric(bandwidths[1,
            k]))
    })), n, n)
```

```
    }
    if (!adapt) {
        res = gwglmnet.fit(x, y, coords, weight.matrix, s, verbose,
            family, weights)
    }
    else {
        res = gwglmnet.adaptive.fit(x, y, coords, weight.matrix,
            s, verbose, family, weights)
    }
    res[["data"]] = data
    res[["response"]] = as.character(formula[[2]])
    res
}
```

---

gwglmnet.adaptive.fit *Use the adaptive LASSO to fit a GLM in the GWR setting.*

Description

Use the adaptive LASSO to fit a GLM in the GWR setting.

Usage

```
gwglmnet.adaptive.fit(x, y, coords, weight.matrix, s, verbose, family, prior.weights)
```

Arguments

x
y
coords
weight.matrix
s
verbose
family
prior.weights

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y, coords, weight.matrix, s, verbose, family, prior.weights)
{
    gwglmnet.object = list()
    coords.unique = unique(coords)
```

```
model = list()
s.optimal = vector()
adapt.normx = list()
adapt.scale = list()
cv.error = list()
coef.scale = list()
glm.step = list()
for (i in 1:dim(coords.unique)[1]) {
    colocated = which(coords[, 1] == coords.unique[i, 1] &
        coords[, 2] == coords.unique[i, 2])
    loow = weight.matrix[i, -colocated]
    if (sum(loow) == 0) {
        return(list(cv.error = Inf))
    }
    prior.loow = prior.weights[-colocated]
    reps = length(colocated)
    w <- prior.loow * loow
    xx = as.matrix(x[-colocated, ])
    yy = as.matrix(y[-colocated])
    if (family == "binomial" && (abs(sum(yy * w) - sum(w)) <
        1e-04 || sum(yy * w) < 1e-04)) {
        cat(paste("Abort. i=", i, ", weighted sum=", sum(yy *
            w), ", sum of weights=", sum(w), "\n", sep = ""))
        model[[i]] = NULL
        cv.error[[i]] = 0
        s.optimal = c(s.optimal, max(s))
    }
    else {
        m <- ncol(xx)
        n <- nrow(xx)
        one <- rep(1, n)
        meanx <- drop(one %*% xx)/n
        x.centered <- scale(xx, meanx, FALSE)
        normx <- sqrt(drop(one %*% (x.centered^2)))
        adapt.normx[[i]] = normx
        names(normx) <- NULL
        xs = x.centered
        for (k in 1:dim(x.centered)[2]) {
            if (normx[k] != 0) {
              xs[, k] = xs[, k]/normx[k]
            }
            else {
              xs[, k] = rep(0, dim(xs)[1])
              normx[k] = Inf
            }
        }
        out.glm = try(glm(yy ~ xs, family = family, weights = w))
        if (class(out.glm) == "try-error") {
            cat(paste("Had to use the last glm for location ",
              i, "\n", sep = ""))
            glm.step[[i]] = out.glm = glm.step[[i - 1]]
        }
        else {
            glm.step[[i]] = out.glm
        }
        beta.glm = out.glm$coeff[2:(m + 1)]
        adapt.weight = abs(beta.glm)
```

```
            adapt.scale[[i]] = adapt.weight
            for (k in 1:dim(x.centered)[2]) {
                if (!is.na(adapt.weight[k])) {
                  xs[, k] = xs[, k] * adapt.weight[k]
                }
                else {
                  xs[, k] = rep(0, dim(xs)[1])
                  adapt.weight[k] = 0
                }
            }
            coef.scale[[i]] = adapt.weight/normx
            names(coef.scale[[i]]) = sapply(strsplit(names(coef.scale[[i]]),
                "xs"), function(x) {
                x[2]
            })
            if (sum(coef.scale[[i]]) < 1e-10) {
                if (verbose) {
                  cat(paste("opted for the intercept-only model at location: ",
                    i, "\n", sep = ""))
                }
                model[[i]] = NULL
                predictions = rep(coef(out.glm)[["(Intercept)"]],
                  length(colocated))
                cv.error[[i]] = abs(matrix(predictions - matrix(y[colocated],
                  nrow = reps, ncol = length(s))))
                s.optimal = c(s.optimal, max(s))
            }
            else {
                if (family == "binomial") {
                  model[[i]] = glmnet(x = xs, y = cbind(1 - yy,
                    yy), lambda = s, family = family, weights = w)
                }
                else {
                  model[[i]] = glmnet(x = xs, y = yy, lambda = s,
                    family = family, weights = w)
                }
                predictions = predict(model[[i]], newx = scale(matrix(x[colocated,
                  ], nrow = reps, ncol = dim(xx)[2]), center = meanx,
                  scale = normx/adapt.weight), type = "response",
                  s = s)
                cv.error[[i]] = colSums(abs(matrix(predictions -
                  matrix(y[colocated], nrow = reps, ncol = length(s)),
                  nrow = reps, ncol = length(s))))
                s.optimal = c(s.optimal, s[which.min(cv.error[[i]])])
            }
        }
        if (verbose) {
            cat(paste(i, "\n", sep = ""))
        }
    }
    gwglmnet.object[["coef.scale"]] = coef.scale
    gwglmnet.object[["model"]] = model
    gwglmnet.object[["s"]] = s.optimal
    gwglmnet.object[["mode"]] = mode
    gwglmnet.object[["coords"]] = coords.unique
    gwglmnet.object[["cv.error"]] = cv.error
    gwglmnet.object[["s.range"]] = s
```

```
  class(gwglmnet.object) = "gwglmnet.object"
  return(gwglmnet.object)
}
```

---

gwglmnet.adaptive.ssr *Get the sum of squared residuals in for a geographically-weighted GLM*

Description

Get the sum of squared residuals in for a geographically-weighted GLM

Usage

```
gwglmnet.adaptive.ssr(bw, x, y, colocated, dist, s, verbose, family, prior.weights, gweight, typ
```

Arguments

bw

x

y

colocated

dist

s

verbose

family

prior.weights

gweight

type

target

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (bw, x, y, colocated, dist, s, verbose, family, prior.weights,
    gweight, type, target)
{
    reps = length(colocated)
    loow = gweight(dist, bw)[-colocated]
    w <- prior.weights[-colocated] * loow
    xx = as.matrix(x[-colocated, ])
    yy = as.matrix(y[-colocated])
```

```
m <- ncol(xx)
n <- nrow(xx)
one <- rep(1, n)
meanx <- drop(one %*% xx)/n
x.centered <- scale(xx, meanx, FALSE)
normx <- sqrt(drop(one %*% (x.centered^2)))
names(normx) <- NULL
xs = x.centered
for (k in 1:dim(x.centered)[2]) {
    if (normx[k] != 0) {
        xs[, k] = xs[, k]/normx[k]
    }
    else {
        xs[, k] = rep(0, dim(xs)[1])
        normx[k] = Inf
    }
}
glm.step = try(glm(yy ~ xs, family = family, weights = w))
if (class(glm.step) == "try-error") {
    cat(paste("Couldn't make a model for finding the SSR at bandwidth ",
        bw, "\n", sep = ""))
    return(Inf)
}
beta.glm = glm.step$coeff[2:(m + 1)]
adapt.weight = abs(beta.glm)
for (k in 1:dim(x.centered)[2]) {
    if (!is.na(adapt.weight[k])) {
        xs[, k] = xs[, k] * adapt.weight[k]
    }
    else {
        xs[, k] = rep(0, dim(xs)[1])
        adapt.weight[k] = 0
    }
}
if (family == "binomial") {
    model = glmnet(x = xs, y = cbind(1 - yy, yy), weights = w,
        family = family, lambda = s)
}
else {
    model = glmnet(x = xs, y = yy, weights = w, family = family,
        lambda = s)
}
ll = model$lambda
xs.colocated = (x[colocated, ] - meanx) * adapt.weight/normx
predictions = predict(model, newx = matrix(xs.colocated,
    nrow = reps, ncol = dim(xs)[2]), s = ll, type = "response",
    )
cv.error = colSums(abs(matrix(predictions - matrix(y[colocated],
    nrow = reps, ncol = length(ll)), nrow = reps, ncol = length(ll))))
s.optimal = ll[which.min(cv.error)]
fitted = predict(model, newx = xs, s = s.optimal, type = "response")
if (family == "poisson")
    pearson.resid = sum(w * (yy - fitted)^2/fitted)
if (family == "binomial")
    pearson.resid = sum(w * (yy - fitted)^2/(fitted * (1 -
        fitted)))
(abs(pearson.resid - target))^2
```

```
}
```

---

gwglmnet.cv.f *Perform cross-validation for bandwidth selection in a gw-glm model.*

Description

Perform cross-validation for bandwidth selection in a gw-glm model.

Usage

```
gwglmnet.cv.f(formula, data, bw, coords, gweight, verbose, adapt, longlat, s, family, weights, n
```

Arguments

formula
data
bw
coords
gweight
verbose
adapt
longlat
s
family
weights
nn
D
...

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data, bw, coords, gweight, verbose, adapt,
    longlat, s, family, weights, nn, D = NULL, ...)
{
    cat(paste("Beginning with bandwidth: ", bw, "\n", sep = ""))
    gwglmnet.model = gwglmnet(formula = formula, data = data,
        coords = coords, gweight = gweight, bw = bw, verbose = verbose,
        longlat = longlat, adapt = adapt, s = s, family = family,
        weights = weights, nearest.neighbors = nn, D = D)
```

```
    cv.error = sum(sapply(gwglmnet.model[["cv.error"]], min))
    cat(paste("Bandwidth: ", bw, ". CV error: ", cv.error, "\n",
        sep = ""))
    return(cv.error)
  }
```

---

gwglmnet.fit *Fit a gw-glm model using the LASSO for variable selection*

Description

Fit a gw-glm model using the LASSO for variable selection

Usage

```
gwglmnet.fit(x, y, coords, weight.matrix, s, verbose, family, prior.weights)
```

Arguments

x
y
coords
weight.matrix
s
verbose
family
prior.weights

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y, coords, weight.matrix, s, verbose, family, prior.weights)
{
    coords.unique = unique(coords)
    model = list()
    s.optimal = vector()
    gwglmnet.object = list()
    cv.error = list()
    for (i in 1:dim(coords.unique)[1]) {
        colocated = which(coords[, 1] == coords.unique[i, 1] &
            coords[, 2] == coords.unique[i, 2])
        loow = weight.matrix[i, -colocated]
        prior.loow = prior.weights[-colocated]
```

```
        reps = length(colocated)
        w <- prior.loow * loow
        if (sum(loow) == 0) {
            return(list(cv.error = Inf))
        }
        reps = length(colocated)
        xx = as.matrix(x[-colocated, ])
        yy = as.matrix(y[-colocated])
        if (family == "binomial" && (abs(sum(yy * w) - sum(w)) <
            1e-04 || sum(yy * w) < 1e-04)) {
            cat(paste("Abort. i=", i, ", weighted sum=", sum(yy *
                w), ", sum of weights=", sum(w), "\n", sep = ""))
            model[[i]] = NULL
            cv.error[[i]] = 0
            s.optimal = c(s.optimal, max(s))
        }
        else {
            model[[i]] = glmnet(x = xx, y = cbind(1 - yy, yy),
                weights = w, family = family, lambda = s)
            predictions = predict(model[[i]], newx = matrix(x[colocated,
                ], nrow = reps, ncol = dim(xx)[2]), s = s, type = "response")
            cv.error[[i]] = colSums(abs(matrix(predictions -
                matrix(y[colocated], nrow = reps, ncol = length(s)),
                nrow = reps, ncol = length(s))))
            s.optimal = c(s.optimal, s[which.min(cv.error[[i]])])
        }
        if (verbose) {
            cat(paste(i, "\n", sep = ""))
        }
    }
    gwglmnet.object[["coef.scale"]] = NULL
    gwglmnet.object[["model"]] = model
    gwglmnet.object[["s"]] = s.optimal
    gwglmnet.object[["mode"]] = mode
    gwglmnet.object[["coords"]] = coords.unique
    gwglmnet.object[["cv.error"]] = cv.error
    gwglmnet.object[["s.range"]] = s
    class(gwglmnet.object) = "gwglmnet.object"
    return(gwglmnet.object)
}
```

`gwglmnet.nen` *Create a GW-GLM model using the LASSO for variable selection and Nearest Effective Neighbors for bandwidth selection.*

Description

Create a GW-GLM model using the LASSO for variable selection and Nearest Effective Neighbors for bandwidth selection.

Usage

```
gwglmnet.nen(formula, data, coords, gweight, bw, D = NULL, verbose = FALSE, longlat = FALSE, ada
```

Arguments

formula

data

coords

gweight

bw

D

verbose

longlat

adapt

s

family

weights

tolerance

beta1

beta2

type

parallel

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data, coords, gweight, bw, D = NULL, verbose = FALSE,
    longlat = FALSE, adapt = FALSE, s = NULL, family, weights = NULL,
    tolerance = .Machine$double.eps^0.25, beta1 = NULL, beta2 = NULL,
    type, parallel = FALSE)
{
    if (!is.logical(adapt))
        stop("adapt must be logical")
    if (is(data, "Spatial")) {
        if (!missing(coords))
            warning("data is Spatial* object, ignoring coords argument")
        coords <- coordinates(data)
        if ((is.null(longlat) || !is.logical(longlat)) && !is.na(is.projected(data)) &&
            !is.projected(data)) {
            longlat <- TRUE
        }
        else longlat <- FALSE
        data <- as(data, "data.frame")
    }
    if (is.null(longlat) || !is.logical(longlat))
```

```
    longlat <- FALSE
if (missing(coords))
    stop("Observation coordinates have to be given")
mf <- match.call(expand.dots = FALSE)
m <- match(c("formula", "data"), names(mf), 0)
mf <- mf[c(1, m)]
mf$drop.unused.levels <- TRUE
mf[[1]] <- as.name("model.frame")
mf <- eval(mf, parent.frame())
mt <- attr(mf, "terms")
dp.n <- length(model.extract(mf, "response"))
if (!is.null(weights) && !is.numeric(weights))
    stop("'weights' must be a numeric vector")
if (is.null(weights))
    weights <- rep(as.numeric(1), dp.n)
if (any(is.na(weights)))
    stop("NAs in weights")
if (any(weights < 0))
    stop("negative weights")
y <- model.extract(mf, "response")
x <- model.matrix(mt, mf)
if (is.null(D)) {
    n = dim(coords)[1]
    if (longlat) {
        D = as.matrix(earth.dist(coords), n, n)
    }
    else {
        Xmat = matrix(rep(coords[, 1], times = n), n, n)
        Ymat = matrix(rep(coords[, 2], times = n), n, n)
        D = sqrt((Xmat - t(Xmat))^2 + (Ymat - t(Ymat))^2)
    }
}
n = dim(D)[1]
if (is.null(beta1) || is.null(beta2)) {
    bbox <- cbind(range(coords[, 1]), range(coords[, 2]))
    difmin <- spDistsN1(bbox, bbox[2, ], longlat)[1]
    if (any(!is.finite(difmin)))
        difmin[which(!is.finite(difmin))] <- 0
    beta1 <- difmin/1000
    beta2 <- 2 * difmin
}
res = list()
if (adapt) {
    if (parallel) {
        res[["model"]] = gwglmnet.nen.adaptive.fit.parallel(x,
            y, coords, D, s, verbose, family, weights, gweight,
            bw, beta1, beta2, type, tol = toelrance, longlat = longlat)
    }
    else {
        res[["model"]] = gwglmnet.nen.adaptive.fit(x, y,
            coords, D, s, verbose, family, weights, gweight,
            bw, beta1, beta2, type, tol = toelrance, longlat = longlat)
    }
}
if (!adapt) {
    if (!parallel) {
        res[["model"]] = gwglmnet.nen.fit(x, y, coords, D,
```

```
                s, verbose, family, weights, gweight, bw, beta1,
                beta2, type = type, tol = tolerance, longlat = longlat)
        }
        else {
            res[["model"]] = gwglmnet.nen.fit.parallel(x, y,
                coords, D, s, verbose, family, weights, gweight,
                bw, beta1, beta2, type = type, tol = tolerance,
                longlat = longlat)
        }
    }
    res[["data"]] = data
    res[["response"]] = as.character(formula[[2]])
    res
  }
```

---

gwglmnet.nen.adaptive.fit *Fit a GW-GLM model using the LASSO for variable selection and Nearest Effective Neighbors for bandwidth selection.*

Description

Fit a GW-GLM model using the LASSO for variable selection and Nearest Effective Neighbors for bandwidth selection.

Usage

```
gwglmnet.nen.adaptive.fit.parallel(x, y, coords, D, s, verbose, family, prior.weights, gweight,
```

Arguments

x

y

coords

D

s

verbose

family

prior.weights

gweight

target

beta1

beta2

type

tol

longlat

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y, coords, D, s, verbose, family, prior.weights,
    gweight, target, beta1, beta2, type = "pearson", tol = 1e-25,
    longlat = FALSE)
{
    coords.unique = unique(coords)
    n = dim(coords.unique)[1]
    gwglmnet.object = foreach(i = 1:n, .packages = "glmnet",
        .errorhandling = "remove") %dopar% {
        colocated = which(coords[, 1] == coords.unique[i, 1] &
            coords[, 2] == coords.unique[i, 2])
        dist = D[i, ]
        opt = optimize(gwglmnet.adaptive.ssr, lower = beta1,
            upper = beta2, maximum = FALSE, tol = target/1000,
            x = x, y = y, colocated = colocated, s = s, gweight = gweight,
            verbose = verbose, dist = dist, prior.weights = prior.weights,
            family = family, target = target, type = type)
        bandwidth = opt$minimum
        cat(paste("For i=", i, ", bw=", bandwidth, ", tolerance=",
            target/1000, ", miss=", sqrt(opt$objective), ".\n",
            sep = ""))
        loow = gweight(D[i, -colocated], bandwidth)
        prior.loow = prior.weights[-colocated]
        w <- prior.loow * loow
        reps = length(colocated)
        if (sum(loow) == 0) {
            return(list(cv.error = Inf))
        }
        xx = as.matrix(x[-colocated, ])
        yy = as.matrix(y[-colocated])
        m <- ncol(xx)
        n <- nrow(xx)
        one <- rep(1, n)
        meanx <- drop(one %*% xx)/n
        x.centered <- scale(xx, meanx, FALSE)
        normx <- sqrt(drop(one %*% (x.centered^2)))
        names(normx) <- NULL
        xs = x.centered
        for (k in 1:dim(x.centered)[2]) {
            if (normx[k] != 0) {
                xs[, k] = xs[, k]/normx[k]
            }
            else {
                xs[, k] = rep(0, dim(xs)[1])
                normx[k] = Inf
            }
        }
```

```
glm.step = try(glm(yy ~ xs, family = family, weights = w))
if (class(out.glm) == "try-error") {
    cat(paste("Couldn't make a model for finding the SSR at location ",
        i, ", bandwidth ", bw, "\n", sep = ""))
    return(Inf)
}
beta.glm = glm.step$coeff[2:(m + 1)]
adapt.weight = abs(beta.glm)
for (k in 1:dim(x.centered)[2]) {
    if (!is.na(adapt.weight[k])) {
        xs[, k] = xs[, k] * adapt.weight[k]
    }
    else {
        xs[, k] = rep(0, dim(xs)[1])
        adapt.weight[k] = 0
    }
}
print(family)
print(sum(yy * w))
if (family == "binomial" && (abs(sum(yy * w) - sum(w)) <
    1e-04 || sum(yy * w) < 1e-04)) {
    cat(paste("Abort. i=", i, ", weighted sum=", sum(yy *
        w), ", sum of weights=", sum(w), "\n", sep = ""))
    model = NULL
    cv.error = 0
    s.optimal = max(s)
}
else if (family == "binomial") {
    print("Right choice")
    xs.colocated = (x[colocated, ] - meanx) * adapt.weight/normx
    model = glmnet(x = xs, y = cbind(1 - yy, yy), weights = w,
        family = family, lambda = s)
    predictions = predict(model, newx = matrix(xs.colocated,
        nrow = reps, ncol = dim(xx)[2]), s = ll, type = "response")
    cv.error = colSums(abs(matrix(predictions - matrix(y[colocated],
        nrow = reps, ncol = length(s)), nrow = reps,
        ncol = length(s))))
    s.optimal = ll[which.min(cv.error)]
    print(cv.error)
}
else {
    xs.colocated = (x[colocated, ] - meanx) * adapt.weight/normx
    model = glmnet(x = xs, y = yy, weights = w, family = family,
        lambda = s)
    ll = model$lambda
    predictions = predict(model, newx = matrix(xs.colocated,
        nrow = reps, ncol = dim(xx)[2]), s = ll, type = "response")
    cv.error = colSums(abs(matrix(predictions - matrix(y[colocated],
        nrow = reps, ncol = length(ll)), nrow = reps,
        ncol = length(ll))))
    s.optimal = ll[which.min(cv.error)]
}
if (verbose) {
    cat(paste(i, "\n", sep = ""))
}
list(model = model, cv.error = cv.error, s = s.optimal,
    index = i)
```

```
  }
  print("returning from gwglmnet.nen.adaptive.fit.parallel")
  class(gwglmnet.object) = "gwglmnet.object"
  return(gwglmnet.object)
}
```

---

gwglmnet.nen.adaptive.fit.parallel

Fit a GW-GLM model using the LASSO for variable selection and Nearest Effective Neighbors for bandwidth selection.

Description

Fit a GW-GLM model using the LASSO for variable selection and Nearest Effective Neighbors for bandwidth selection.

Usage

```
gwglmnet.nen.adaptive.fit.parallel(x, y, coords, D, s, verbose, family, prior.weights, gweight,
```

Arguments

x

y

coords

D

s

verbose

family

prior.weights

gweight

target

beta1

beta2

type

tol

longlat

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y, coords, D, s, verbose, family, prior.weights,
    gweight, target, beta1, beta2, type = "pearson", tol = 1e-25,
    longlat = FALSE)
{
    coords.unique = unique(coords)
    n = dim(coords.unique)[1]
    gwglmnet.object = foreach(i = 1:n, .packages = "glmnet",
        .errorhandling = "remove") %dopar% {
        colocated = which(coords[, 1] == coords.unique[i, 1] &
            coords[, 2] == coords.unique[i, 2])
        dist = D[i, ]
        opt = optimize(gwglmnet.adaptive.ssr, lower = beta1,
            upper = beta2, maximum = FALSE, tol = target/1000,
            x = x, y = y, colocated = colocated, s = s, gweight = gweight,
            verbose = verbose, dist = dist, prior.weights = prior.weights,
            family = family, target = target, type = type)
        bandwidth = opt$minimum
        cat(paste("For i=", i, ", bw=", bandwidth, ", tolerance=",
            target/1000, ", miss=", sqrt(opt$objective), ".\n",
            sep = ""))
        loow = gweight(D[i, -colocated], bandwidth)
        prior.loow = prior.weights[-colocated]
        w <- prior.loow * loow
        reps = length(colocated)
        if (sum(loow) == 0) {
            return(list(cv.error = Inf))
        }
        xx = as.matrix(x[-colocated, ])
        yy = as.matrix(y[-colocated])
        m <- ncol(xx)
        n <- nrow(xx)
        one <- rep(1, n)
        meanx <- drop(one %*% xx)/n
        x.centered <- scale(xx, meanx, FALSE)
        normx <- sqrt(drop(one %*% (x.centered^2)))
        names(normx) <- NULL
        xs = x.centered
        for (k in 1:dim(x.centered)[2]) {
            if (normx[k] != 0) {
                xs[, k] = xs[, k]/normx[k]
            }
            else {
                xs[, k] = rep(0, dim(xs)[1])
                normx[k] = Inf
            }
        }
        glm.step = try(glm(yy ~ xs, family = family, weights = w))
        if (class(out.glm) == "try-error") {
            cat(paste("Couldn't make a model for finding the SSR at location ",
                i, ", bandwidth ", bw, "\n", sep = ""))
```

```
        return(Inf)
    }
    beta.glm = glm.step$coeff[2:(m + 1)]
    adapt.weight = abs(beta.glm)
    for (k in 1:dim(x.centered)[2]) {
        if (!is.na(adapt.weight[k])) {
            xs[, k] = xs[, k] * adapt.weight[k]
        }
        else {
            xs[, k] = rep(0, dim(xs)[1])
            adapt.weight[k] = 0
        }
    }
    print(family)
    print(sum(yy * w))
    if (family == "binomial" && (abs(sum(yy * w) - sum(w)) <
        1e-04 || sum(yy * w) < 1e-04)) {
        cat(paste("Abort. i=", i, ", weighted sum=", sum(yy *
            w), ", sum of weights=", sum(w), "\n", sep = ""))
        model = NULL
        cv.error = 0
        s.optimal = max(s)
    }
    else if (family == "binomial") {
        print("Right choice")
        xs.colocated = (x[colocated, ] - meanx) * adapt.weight/normx
        model = glmnet(x = xs, y = cbind(1 - yy, yy), weights = w,
            family = family, lambda = s)
        predictions = predict(model, newx = matrix(xs.colocated,
            nrow = reps, ncol = dim(xx)[2]), s = ll, type = "response")
        cv.error = colSums(abs(matrix(predictions - matrix(y[colocated],
            nrow = reps, ncol = length(s)), nrow = reps,
            ncol = length(s))))
        s.optimal = ll[which.min(cv.error)]
        print(cv.error)
    }
    else {
        xs.colocated = (x[colocated, ] - meanx) * adapt.weight/normx
        model = glmnet(x = xs, y = yy, weights = w, family = family,
            lambda = s)
        ll = model$lambda
        predictions = predict(model, newx = matrix(xs.colocated,
            nrow = reps, ncol = dim(xx)[2]), s = ll, type = "response")
        cv.error = colSums(abs(matrix(predictions - matrix(y[colocated],
            nrow = reps, ncol = length(ll)), nrow = reps,
            ncol = length(ll))))
        s.optimal = ll[which.min(cv.error)]
    }
    if (verbose) {
        cat(paste(i, "\n", sep = ""))
    }
    list(model = model, cv.error = cv.error, s = s.optimal,
        index = i)
}
print("returning from gwglmnet.nen.adaptive.fit.parallel")
class(gwglmnet.object) = "gwglmnet.object"
return(gwglmnet.object)
```

```
}
```

`gwglmnet.nen.cv.f` *Cross-validation for selection of tuning parameter in a GW-GLM model using Nearest Effective Neighbors for bandwidth selection.*

Description

Cross-validation for selection of tuning parameter in a GW-GLM model using Nearest Effective Neighbors for bandwidth selection.

Usage

```
gwglmnet.nen.cv.f(formula, data, bw, coords, gweight, verbose, adapt, longlat, s = NULL, beta1,
```

Arguments

`formula`

`data`

`bw`

`coords`

`gweight`

`verbose`

`adapt`

`longlat`

`s`

`beta1`

`beta2`

`family`

`weights`

`D`

`tolerance`

`type`

`parallel`

`...`

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data, bw, coords, gweight, verbose, adapt,
    longlat, s = NULL, beta1, beta2, family, weights = NULL,
    D = NULL, tolerance = .Machine$double.eps^0.25, type = "pearson",
    parallel = FALSE, ...)
{
    cat(paste("Beginning with target SSR: ", bw, ", tolerance: ",
        tolerance, "\n", sep = ""))
    gwglmnet.model = gwglmnet.nen(formula = formula, data = data,
        coords = coords, gweight = gweight, bw = bw, verbose = verbose,
        longlat = longlat, adapt = adapt, s = s, family = family,
        weights = weights, D = D, tol = tolerance, beta1 = beta1,
        beta2 = beta2, type, parallel = parallel)
    print(gwglmnet.model[["model"]][["cv.error"]])
    print(names(gwglmnet.model))
    print(gwglmnet.model[["model"]])
    cv.error = sum(sapply(gwglmnet.model[["model"]], function(x) min(x[["cv.error"]])))
    cat(paste("Bandwidth: ", bw, ". CV error: ", cv.error, "\n",
        sep = ""))
    return(cv.error)
  }
```

`gwglmnet.nen.fit` *Fit a GW-GLM model using Nearest Effective Neighbors for bandwidth selection.*

Description

Fit a GW-GLM model using Nearest Effective Neighbors for bandwidth selection.

Usage

```
gwglmnet.nen.fit(x, y, coords, D, s = NULL, verbose, family, prior.weights, gweight, bw, beta1,
```

Arguments

`x`

`y`

`coords`

`D`

`s`

`verbose`

`family`

`prior.weights`

`gweight`

bw

beta1

beta2

type

tol

longlat

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y, coords, D, s = NULL, verbose, family, prior.weights,
    gweight, bw, beta1, beta2, type = "pearson", tol = 1e-25,
    longlat = FALSE)
{
    coords.unique = unique(coords)
    model = list()
    s.optimal = vector()
    gwglmnet.object = list()
    cv.error = list()
    for (i in 1:dim(coords.unique)[1]) {
        colocated = which(coords[, 1] == coords.unique[i, 1] &
            coords[, 2] == coords.unique[i, 2])
        dist = D[i, ]
        bandwidth = optimize(gwglmnet.ssr, lower = beta1, upper = beta2,
            maximum = FALSE, tol = bw/10, x = x, y = y, colocated = colocated,
            s = s, gweight = gweight, verbose = verbose, dist = dist,
            prior.weights = prior.weights, family = family, target = bw,
            type = type)$minimum
        cat(paste("For i=", i, ", bw=", bandwidth, ".\n", sep = ""))
        weight.matrix = gweight(D, bandwidth)
        loow = weight.matrix[i, -colocated]
        prior.loow = prior.weights[-colocated]
        reps = length(colocated)
        w <- prior.loow * loow
        if (sum(loow) == 0) {
            return(list(cv.error = Inf))
        }
        reps = length(colocated)
        xx = as.matrix(x[-colocated, ])
        yy = as.matrix(y[-colocated])
        if (family == "binomial" && (abs(sum(yy * w) - sum(w)) <
            1e-04 || sum(yy * w) < 1e-04)) {
            cat(paste("Abort. i=", i, ", weighted sum=", sum(yy *
                w), ", sum of weights=", sum(w), "\n", sep = ""))
            model[[i]] = NULL
            cv.error[[i]] = 0
            s.optimal = c(s.optimal, max(s))
```

```
        }
        else if (family == "binomial") {
            model[[i]] = glmnet(x = xx, y = cbind(1 - yy, yy),
                weights = w, family = family, lambda = s)
            predictions = predict(model[[i]], newx = matrix(x[colocated,
                ], nrow = reps, ncol = dim(xx)[2]), s = s, type = "response")
            cv.error[[i]] = colSums(abs(matrix(predictions -
                matrix(y[colocated], nrow = reps, ncol = length(s)),
                nrow = reps, ncol = length(s))))
            s.optimal = c(s.optimal, s[which.min(cv.error[[i]])])
        }
        else {
            model[[i]] = glmnet(x = xx, y = yy, weights = w,
                family = family, lambda = s)
            predictions = predict(model[[i]], newx = matrix(x[colocated,
                ], nrow = reps, ncol = dim(xx)[2]), s = s, type = "response")
            cv.error[[i]] = colSums(abs(matrix(predictions -
                matrix(y[colocated], nrow = reps, ncol = length(s)),
                nrow = reps, ncol = length(s))))
            s.optimal = c(s.optimal, s[which.min(cv.error[[i]])])
        }
        if (verbose) {
            cat(paste(i, "\n", sep = ""))
        }
    }
    gwglmnet.object[["coef.scale"]] = NULL
    gwglmnet.object[["model"]] = model
    gwglmnet.object[["s"]] = s.optimal
    gwglmnet.object[["mode"]] = mode
    gwglmnet.object[["coords"]] = coords.unique
    gwglmnet.object[["cv.error"]] = cv.error
    gwglmnet.object[["s.range"]] = s
    class(gwglmnet.object) = "gwglmnet.object"
    return(gwglmnet.object)
}
```

`gwglmnet.nen.fit.parallel` *Multicore-aware function to fit a GW-GLM model using the LASSO for variable selection and Nearest Effective Neighbors for bandwidth selection.*

Description

Multicore-aware function to fit a GW-GLM model using the LASSO for variable selection and Nearest Effective Neighbors for bandwidth selection.

Usage

```
gwglmnet.nen.fit.parallel(x, y, coords, D, s, verbose, family, prior.weights, gweight, target, b
```

Arguments

x

y

coords

D

s

verbose

family

prior.weights

gweight

target

beta1

beta2

type

tol

longlat

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (x, y, coords, D, s, verbose, family, prior.weights,
    gweight, target, beta1, beta2, type = "pearson", tol = 1e-25,
    longlat = FALSE)
{
    coords.unique = unique(coords)
    n = dim(coords.unique)[1]
    gwglmnet.object = foreach(i = 1:n, .packages = "glmnet",
        .errorhandling = "remove") %dopar% {
        colocated = which(coords[, 1] == coords.unique[i, 1] &
            coords[, 2] == coords.unique[i, 2])
        dist = D[i, ]
        opt = optimize(gwglmnet.ssr, lower = beta1, upper = beta2,
            maximum = FALSE, tol = target/1000, x = x, y = y,
            colocated = colocated, s = s, gweight = gweight,
            verbose = verbose, dist = dist, prior.weights = prior.weights,
            family = family, target = target, type = type)
        bandwidth = opt$minimum
        cat(paste("For i=", i, ", bw=", bandwidth, ", tolerance=",
            target/1000, ", miss=", sqrt(opt$objective), ".\n",
            sep = ""))
        loow = gweight(D[i, -colocated], bandwidth)
        prior.loow = prior.weights[-colocated]
```

```
        w <- prior.loow * loow
        reps = length(colocated)
        if (sum(loow) == 0) {
            return(list(cv.error = Inf))
        }
        xx = as.matrix(x[-colocated, ])
        yy = as.matrix(y[-colocated])
        if (family == "binomial" && (abs(sum(yy * w) - sum(w)) <
            1e-04 || sum(yy * w) < 1e-04)) {
            cat(paste("Abort. i=", i, ", weighted sum=", sum(yy *
                w), ", sum of weights=", sum(w), "\n", sep = ""))
            model = NULL
            cv.error = 0
            s.optimal = max(s)
        }
        else if (family == "binomial") {
            model = glmnet(x = xx, y = cbind(1 - yy, yy), weights = w,
                family = family, lambda = s)
            predictions = predict(model, newx = matrix(x[colocated,
                ], nrow = reps, ncol = dim(xx)[2]), s = s, type = "response")
            cv.error = colSums(abs(matrix(predictions - matrix(y[colocated],
                nrow = reps, ncol = length(s)), nrow = reps,
                ncol = length(s))))
            s.optimal = s[which.min(cv.error)]
        }
        else {
            model = glmnet(x = xx, y = yy, weights = w, family = family,
                lambda = s)
            ll = model$lambda
            predictions = predict(model, newx = matrix(x[colocated,
                ], nrow = reps, ncol = dim(xx)[2]), s = ll, type = "response")
            cv.error = colSums(abs(matrix(predictions - matrix(y[colocated],
                nrow = reps, ncol = length(ll)), nrow = reps,
                ncol = length(ll))))
            s.optimal = ll[which.min(cv.error)]
        }
        if (verbose) {
            cat(paste(i, "\n", sep = ""))
        }
        list(model = model, cv.error = cv.error, s = s.optimal,
            index = i)
    }
    print("returning from gwglmnet.nen.fit.parallel")
    class(gwglmnet.object) = "gwglmnet.object"
    return(gwglmnet.object)
}
```

`gwglmnet.nen.sel` *Bandwidth selection using Nearest Effective Neighbors in a GW-GLM model.*

Description

Bandwidth selection using Nearest Effective Neighbors in a GW-GLM model.

Usage

```
gwglmnet.nen.sel(formula, data = list(), coords, adapt = FALSE, gweight = gwr.Gauss, s = NULL, m
```

Arguments

formula

data

coords

adapt

gweight

s

method

verbose

longlat

family

weights

tol

type

parallel

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data = list(), coords, adapt = FALSE, gweight = gwr.Gauss,
    s = NULL, method = "cv", verbose = FALSE, longlat = FALSE,
    family, weights = NULL, tol = .Machine$double.eps^0.25, type,
    parallel = FALSE)
{
    if (!is.logical(adapt))
        stop("adapt must be logical")
    if (is(data, "Spatial")) {
        if (!missing(coords))
            warning("data is Spatial* object, ignoring coords argument")
        coords <- coordinates(data)
        if ((is.null(longlat) || !is.logical(longlat)) && !is.na(is.projected(data)) &&
            !is.projected(data)) {
            longlat <- TRUE
        }
        else longlat <- FALSE
        data <- as(data, "data.frame")
    }
    if (is.null(longlat) || !is.logical(longlat))
        longlat <- FALSE
```

```
  if (missing(coords))
      stop("Observation coordinates have to be given")
  mf <- match.call(expand.dots = FALSE)
  m <- match(c("formula", "data", "weights"), names(mf), 0)
  mf <- mf[c(1, m)]
  mf$drop.unused.levels <- TRUE
  mf[[1]] <- as.name("model.frame")
  mf <- eval(mf, parent.frame())
  mt <- attr(mf, "terms")
  dp.n <- length(model.extract(mf, "response"))
  if (!is.null(weights) && !is.numeric(weights))
      stop("'weights' must be a numeric vector")
  if (is.null(weights))
      weights <- rep(1, dp.n)
  if (any(is.na(weights)))
      stop("NAs in weights")
  if (any(weights < 0))
      stop("negative weights")
  y <- model.extract(mf, "response")
  x <- model.matrix(mt, mf)
  n = dim(coords)[1]
  if (longlat) {
      D = as.matrix(earth.dist(coords), n, n)
  }
  else {
      Xmat = matrix(rep(coords[, 1], times = n), n, n)
      Ymat = matrix(rep(coords[, 2], times = n), n, n)
      D = sqrt((Xmat - t(Xmat))^2 + (Ymat - t(Ymat))^2)
  }
  model = glm(formula = formula, data = data, family = family,
      weights = weights)
  SSR = sum((weights * residuals(model, type = type))^2)
  cat(paste("The SSR from the global model is: ", SSR, "\n",
      sep = ""))
  nloc = unique(coords)
  lowerSSR <- SSR/5000
  upperSSR <- SSR
  bbox <- cbind(range(coords[, 1]), range(coords[, 2]))
  difmin <- spDistsN1(bbox, bbox[2, ], longlat)[1]
  if (any(!is.finite(difmin)))
      difmin[which(!is.finite(difmin))] <- 0
  beta1 <- difmin/1000
  beta2 <- difmin
  cat(paste("Maximum distance: ", difmin, "\n", sep = ""))
  opt <- optimize(gwglmnet.nen.cv.f, lower = lowerSSR, upper = upperSSR,
      maximum = FALSE, tol = tol, tolerance = tol, formula = formula,
      coords = coords, s = s, beta1 = beta1, beta2 = beta2,
      gweight = gweight, verbose = verbose, longlat = longlat,
      data = data, D = D, weights = weights, adapt = adapt,
      family = family, type = type, parallel = parallel)
  bdwt <- opt$minimum
  res <- bdwt
  res
}
```

gwglmnet.sel *Bandwidth selection in a GW-GLM model (bandwidth in terms of nearest neighbors or distance).*

Description

Bandwidth selection in a GW-GLM model (bandwidth in terms of nearest neighbors or distance).

Usage

```
gwglmnet.sel(formula, data = list(), coords, adapt = FALSE, nearest.neighbors = FALSE, gweight =
```

Arguments

formula

data

coords

adapt

nearest.neighbors

gweight

s

method

verbose

longlat

family

weights

tol

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data = list(), coords, adapt = FALSE, nearest.neighbors = FALSE,
    gweight = gwr.Gauss, s, method = "cv", verbose = FALSE, longlat = FALSE,
    family, weights, tol = .Machine$double.eps^0.25)
{
    if (!is.logical(adapt))
        stop("adapt must be logical")
    if (is(data, "Spatial")) {
        if (!missing(coords))
            warning("data is Spatial* object, ignoring coords argument")
```

```
    coords <- coordinates(data)
    if ((is.null(longlat) || !is.logical(longlat)) && !is.na(is.projected(data)) &&
        !is.projected(data)) {
        longlat <- TRUE
    }
    else longlat <- FALSE
    data <- as(data, "data.frame")
}
if (is.null(longlat) || !is.logical(longlat))
    longlat <- FALSE
if (missing(coords))
    stop("Observation coordinates have to be given")
mf <- match.call(expand.dots = FALSE)
m <- match(c("formula", "data", "weights"), names(mf), 0)
mf <- mf[c(1, m)]
mf$drop.unused.levels <- TRUE
mf[[1]] <- as.name("model.frame")
mf <- eval(mf, parent.frame())
mt <- attr(mf, "terms")
dp.n <- length(model.extract(mf, "response"))
weights <- as.vector(model.extract(mf, "weights"))
if (!is.null(weights) && !is.numeric(weights))
    stop("'weights' must be a numeric vector")
if (is.null(weights))
    weights <- rep(as.numeric(1), dp.n)
if (any(is.na(weights)))
    stop("NAs in weights")
if (any(weights < 0))
    stop("negative weights")
y <- model.extract(mf, "response")
x <- model.matrix(mt, mf)
n = dim(coords)[1]
if (longlat) {
    D = as.matrix(earth.dist(coords), n, n)
}
else {
    Xmat = matrix(rep(coords[, 1], times = n), n, n)
    Ymat = matrix(rep(coords[, 2], times = n), n, n)
    D = sqrt((Xmat - t(Xmat))^2 + (Ymat - t(Ymat))^2)
}
if (nearest.neighbors) {
    beta1 <- 0
    beta2 <- 1
}
else {
    bbox <- cbind(range(coords[, 1]), range(coords[, 2]))
    difmin <- spDistsN1(bbox, bbox[2, ], longlat)[1]
    if (any(!is.finite(difmin)))
        difmin[which(!is.finite(difmin))] <- 0
    beta1 <- difmin/1000
    beta2 <- 2 * difmin
}
opt <- optimize(gwglmnet.cv.f, lower = beta1, upper = beta2,
    maximum = FALSE, tol = tol, formula = formula, coords = coords,
    s = s, gweight = gweight, verbose = verbose, longlat = longlat,
    data = data, D = D, weights = weights, adapt = adapt,
    nn = nearest.neighbors, family = family)
```

```
    bdwt <- opt$minimum
    res <- bdwt
    res
  }
```

---

gwr.matplot *Heatmap plotting function for gwrselect package*

Description

Heatmap plotting function for gwrselect package

Usage

```
gwr.matplot()
```

Arguments

formula

data

coords

gweight

bw

D

verbose

longlat

adapt

s

family

weights

nearest.neighbors

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data, coords, gweight, bw, D = NULL, verbose = FALSE,
    longlat = FALSE, adapt = FALSE, s, family, weights = NULL,
    nearest.neighbors = FALSE)
{
    if (!is.logical(adapt))
```

```
    stop("adapt must be logical")
if (is(data, "Spatial")) {
    if (!missing(coords))
        warning("data is Spatial* object, ignoring coords argument")
    coords <- coordinates(data)
    if ((is.null(longlat) || !is.logical(longlat)) && !is.na(is.projected(data)) &&
        !is.projected(data)) {
        longlat <- TRUE
    }
    else longlat <- FALSE
    data <- as(data, "data.frame")
}
if (is.null(longlat) || !is.logical(longlat))
    longlat <- FALSE
if (missing(coords))
    stop("Observation coordinates have to be given")
mf <- match.call(expand.dots = FALSE)
m <- match(c("formula", "data"), names(mf), 0)
mf <- mf[c(1, m)]
mf$drop.unused.levels <- TRUE
mf[[1]] <- as.name("model.frame")
mf <- eval(mf, parent.frame())
mt <- attr(mf, "terms")
dp.n <- length(model.extract(mf, "response"))
if (!is.null(weights) && !is.numeric(weights))
    stop("'weights' must be a numeric vector")
if (is.null(weights))
    weights <- rep(as.numeric(1), dp.n)
if (any(is.na(weights)))
    stop("NAs in weights")
if (any(weights < 0))
    stop("negative weights")
y <- model.extract(mf, "response")
x <- model.matrix(mt, mf)
if (is.null(D)) {
    n = dim(coords)[1]
    if (longlat) {
        D = as.matrix(earth.dist(coords), n, n)
    }
    else {
        Xmat = matrix(rep(coords[, 1], times = n), n, n)
        Ymat = matrix(rep(coords[, 2], times = n), n, n)
        D = sqrt((Xmat - t(Xmat))^2 + (Ymat - t(Ymat))^2)
    }
}
if (!nearest.neighbors) {
    weight.matrix = gweight(D, bw)
}
else {
    n = dim(D)[1]
    bandwidths = sapply(1:n, function(x) {
        neighbor.weight(q = bw, D = D[x, ], weight.function = gweight,
            verbose = verbose, tol = 0.001)
    })
    weight.matrix = as.matrix(rbind(sapply(1:n, function(k) {
        gweight(as.vector(D[k, ]), as.numeric(bandwidths[1,
            k]))
```

```
        })), n, n)
    }
    if (!adapt) {
        res = gwglmnet.fit(x, y, coords, weight.matrix, s, verbose,
            family, weights)
    }
    else {
        res = gwglmnet.adaptive.fit(x, y, coords, weight.matrix,
            s, verbose, family, weights)
    }
    res[["data"]] = data
    res[["response"]] = as.character(formula[[2]])
    res
}
```

---

registerCores *Register multiple cores for parallelization via doMC*

Description

Register multiple cores for parallelization via doMC

Usage

```
gwglmnet(formula, data, coords, gweight, bw, D = NULL, verbose = FALSE, longlat = FALSE, adapt =
```

Arguments

formula

data

coords

gweight

bw

D

verbose

longlat

adapt

s

family

weights

nearest.neighbors

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data, coords, gweight, bw, D = NULL, verbose = FALSE,
    longlat = FALSE, adapt = FALSE, s, family, weights = NULL,
    nearest.neighbors = FALSE)
{
    if (!is.logical(adapt))
        stop("adapt must be logical")
    if (is(data, "Spatial")) {
        if (!missing(coords))
            warning("data is Spatial* object, ignoring coords argument")
        coords <- coordinates(data)
        if ((is.null(longlat) || !is.logical(longlat)) && !is.na(is.projected(data)) &&
            !is.projected(data)) {
            longlat <- TRUE
        }
        else longlat <- FALSE
        data <- as(data, "data.frame")
    }
    if (is.null(longlat) || !is.logical(longlat))
        longlat <- FALSE
    if (missing(coords))
        stop("Observation coordinates have to be given")
    mf <- match.call(expand.dots = FALSE)
    m <- match(c("formula", "data"), names(mf), 0)
    mf <- mf[c(1, m)]
    mf$drop.unused.levels <- TRUE
    mf[[1]] <- as.name("model.frame")
    mf <- eval(mf, parent.frame())
    mt <- attr(mf, "terms")
    dp.n <- length(model.extract(mf, "response"))
    if (!is.null(weights) && !is.numeric(weights))
        stop("'weights' must be a numeric vector")
    if (is.null(weights))
        weights <- rep(as.numeric(1), dp.n)
    if (any(is.na(weights)))
        stop("NAs in weights")
    if (any(weights < 0))
        stop("negative weights")
    y <- model.extract(mf, "response")
    x <- model.matrix(mt, mf)
    if (is.null(D)) {
        n = dim(coords)[1]
        if (longlat) {
            D = as.matrix(earth.dist(coords), n, n)
        }
        else {
            Xmat = matrix(rep(coords[, 1], times = n), n, n)
            Ymat = matrix(rep(coords[, 2], times = n), n, n)
            D = sqrt((Xmat - t(Xmat))^2 + (Ymat - t(Ymat))^2)
        }
    }
```

```
    if (!nearest.neighbors) {
        weight.matrix = gweight(D, bw)
    }
    else {
        n = dim(D)[1]
        bandwidths = sapply(1:n, function(x) {
            neighbor.weight(q = bw, D = D[x, ], weight.function = gweight,
                verbose = verbose, tol = 0.001)
        })
        weight.matrix = as.matrix(rbind(sapply(1:n, function(k) {
            gweight(as.vector(D[k, ]), as.numeric(bandwidths[1,
                k]))
        })), n, n)
    }
    if (!adapt) {
        res = gwglmnet.fit(x, y, coords, weight.matrix, s, verbose,
            family, weights)
    }
    else {
        res = gwglmnet.adaptive.fit(x, y, coords, weight.matrix,
            s, verbose, family, weights)
    }
    res[["data"]] = data
    res[["response"]] = as.character(formula[[2]])
    res
}
```

---

utils *utility functions for the gwselect package*

Description

utility functions for the gwselect package

Usage

```
gwglmnet(formula, data, coords, gweight, bw, D = NULL, verbose = FALSE, longlat = FALSE, adapt =
```

Arguments

formula

data

coords

gweight

bw

D

verbose

longlat

adapt

s

family

weights

nearest.neighbors

Author(s)

Wesley Brooks

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (formula, data, coords, gweight, bw, D = NULL, verbose = FALSE,
    longlat = FALSE, adapt = FALSE, s, family, weights = NULL,
    nearest.neighbors = FALSE)
{
    if (!is.logical(adapt))
        stop("adapt must be logical")
    if (is(data, "Spatial")) {
        if (!missing(coords))
            warning("data is Spatial* object, ignoring coords argument")
        coords <- coordinates(data)
        if ((is.null(longlat) || !is.logical(longlat)) && !is.na(is.projected(data)) &&
            !is.projected(data)) {
            longlat <- TRUE
        }
        else longlat <- FALSE
        data <- as(data, "data.frame")
    }
    if (is.null(longlat) || !is.logical(longlat))
        longlat <- FALSE
    if (missing(coords))
        stop("Observation coordinates have to be given")
    mf <- match.call(expand.dots = FALSE)
    m <- match(c("formula", "data"), names(mf), 0)
    mf <- mf[c(1, m)]
    mf$drop.unused.levels <- TRUE
    mf[[1]] <- as.name("model.frame")
    mf <- eval(mf, parent.frame())
    mt <- attr(mf, "terms")
    dp.n <- length(model.extract(mf, "response"))
    if (!is.null(weights) && !is.numeric(weights))
        stop("'weights' must be a numeric vector")
    if (is.null(weights))
        weights <- rep(as.numeric(1), dp.n)
    if (any(is.na(weights)))
        stop("NAs in weights")
    if (any(weights < 0))
        stop("negative weights")
    y <- model.extract(mf, "response")
    x <- model.matrix(mt, mf)
    if (is.null(D)) {
```

```
        n = dim(coords)[1]
        if (longlat) {
            D = as.matrix(earth.dist(coords), n, n)
        }
        else {
            Xmat = matrix(rep(coords[, 1], times = n), n, n)
            Ymat = matrix(rep(coords[, 2], times = n), n, n)
            D = sqrt((Xmat - t(Xmat))^2 + (Ymat - t(Ymat))^2)
        }
    }
    if (!nearest.neighbors) {
        weight.matrix = gweight(D, bw)
    }
    else {
        n = dim(D)[1]
        bandwidths = sapply(1:n, function(x) {
            neighbor.weight(q = bw, D = D[x, ], weight.function = gweight,
                verbose = verbose, tol = 0.001)
        })
        weight.matrix = as.matrix(rbind(sapply(1:n, function(k) {
            gweight(as.vector(D[k, ]), as.numeric(bandwidths[1,
                k]))
        })), n, n)
    }
    if (!adapt) {
        res = gwglmnet.fit(x, y, coords, weight.matrix, s, verbose,
            family, weights)
    }
    else {
        res = gwglmnet.adaptive.fit(x, y, coords, weight.matrix,
            s, verbose, family, weights)
    }
    res[["data"]] = data
    res[["response"]] = as.character(formula[[2]])
    res
}
```

Index