Package ‘fastcpd’

September 6, 2023

Type Package

Title Fast Change Point Detection Based on Dynamic Programming with Pruning with Sequential Gradient Descent

Version 0.5.3

Description Implements fast change point detection algorithm based on the paper ``Sequential Gradient Descent and Quasi-Newton's Method for Change-Point Analysis'' by Xianyang Zhang, Trisha Dawn <https://proceedings.mlr.press/v206/zhang23b.html>.

License MIT + file LICENSE

Encoding UTF-8

Imports DescTools,
fastglm,
ggplot2,
glmnet,
Matrix,
methods,
Rcpp (>= 0.11.0),
stats

LinkingTo Rcpp, RcppArmadillo, testthat

RoxygenNote 7.2.3

Suggests abind,
knitr,
mvtnorm,
rmarkdown,
testthat (>= 3.0.0),
xml2

Config/testthat/edition 3

URL https://fastcpd.xingchi.li

R topics documented:

fastcpd — *fastcpd: A package for finding change points in an efficient way*

Description

The fastcpd package provides a function fastcpd to find change points in a data set. The function is based on the paper "Sequential Gradient Descent and Quasi-Newton's Method for Change-Point Analysis" by Xianyang Zhang and Trisha Dawn.

Usage

```
fastcpd(
  formula = y ~ . - 1,
  data,
  beta = NULL,
  segment_count = 10,
  trim = 0.025,
  momentum_coef = 0,
  k = function(x) 0,
  family = NULL,
  epsilon = 1e-10,
  min_prob = 10^10,
  winsorise_minval = -20,
  winsorise_maxval = 20,
  p = NULL,
  cost = negative_log_likelihood,
  cost_gradient = cost_update_gradient,
  cost_hessian = cost_update_hessian,
  cp_only = FALSE,
  vanilla_percentage = 0
)
```

Arguments

formula	A symbolic description of the model to be fitted.
data	A data frame containing the data to be segmented.
beta	Initial cost value.
segment_count	Number of segments for initial guess.
trim	Trimming for the boundary change points.
momentum_coef	Momentum coefficient to be applied to each update.
k	Function on number of epochs in SGD.
family	Family of the models. Can be "binomial", "poisson", "lasso" or "gaussian". If not provided, the user must specify the cost function and its gradient (and Hessian).
epsilon	Epsilon to avoid numerical issues. Only used for binomial and poisson.
min_prob	Minimum probability to avoid numerical issues. Only used for poisson.

winsorise_minval
: Minimum value to be winsorised. Only used for poisson.

winsorise_maxval
: Maximum value to be winsorised. Only used for poisson.

p
: Number of parameters to be estimated.

cost
: Cost function to be used. If not specified, the default is the negative log-likelihood for the corresponding family.

cost_gradient
: Gradient for custom cost function.

cost_hessian
: Hessian for custom cost function.

cp_only
: Whether to return only the change points or with the cost values for each segment. If family is not provided or set to be "custom", this parameter will be set to be true.

vanilla_percentage
: How many of the data should be processed through vanilla PELT. Range should be between 0 and 1. If set to be 0, all data will be processed through sequential gradient descnet. If set to be 1, all data will be processed through vaniall PELT. If the cost function have an explicit solution, i.e. does not depend on coefficients like the mean change case, this parameter will be set to be 1.

Value

A class `fastcpd` object.

Citation

Zhang, Xianyang, and Trisha Dawn. "Sequential Gradient Descent and Quasi-Newton's Method for Change-Point Analysis" arXiv preprint arXiv:2210.12235 (2022).

Examples

```
# Linear regression
library(fastcpd)
set.seed(1)
p <- 3
x <- mvtnorm::rmvnorm(300, rep(0, p), diag(p))
theta_0 <- rbind(c(1, 1.2, -1), c(-1, 0, 0.5), c(0.5, -0.3, 0.2))
y <- c(
  x[1:100, ] %*% theta_0[1, ] + rnorm(100, 0, 1),
  x[101:200, ] %*% theta_0[2, ] + rnorm(100, 0, 1),
  x[201:300, ] %*% theta_0[3, ] + rnorm(100, 0, 1)
)
result <- fastcpd(
  formula = y ~ . - 1,
  data = data.frame(y = y, x = x),
  family = "gaussian"
)
plot(result)
summary(result)

# Linear regression with one-dimensional covariate
library(fastcpd)
set.seed(1)
p <- 1
x <- mvtnorm::rmvnorm(300, rep(0, p), diag(p))
```

```
theta_0 <- matrix(c(1, -1, 0.5))
y <- c(
  x[1:100, ] * theta_0[1, ] + rnorm(100, 0, 1),
  x[101:200, ] * theta_0[2, ] + rnorm(100, 0, 1),
  x[201:300, ] * theta_0[3, ] + rnorm(100, 0, 1)
)
result <- fastcpd(
  formula = y ~ . - 1,
  data = data.frame(y = y, x = x),
  family = "gaussian"
)
plot(result)
summary(result)

# Logistic regression
library(fastcpd)
set.seed(1)
x <- matrix(rnorm(1500, 0, 1), ncol = 5)
theta <- rbind(rnorm(5, 0, 1), rnorm(5, 2, 1))
y <- c(
  rbinom(125, 1, 1 / (1 + exp(-x[1:125, ] %*% theta[1, ]))),
  rbinom(175, 1, 1 / (1 + exp(-x[126:300, ] %*% theta[2, ])))
)
result <- suppressWarnings(fastcpd(
  formula = y ~ . - 1,
  data = data.frame(y = y, x = x),
  family = "binomial"
))
summary(result)

# Poisson regression
library(fastcpd)
set.seed(1)
p <- 3
x <- mvtnorm::rmvnorm(1500, rep(0, p), diag(p))
delta <- rnorm(p)
theta_0 <- c(1, 1.2, -1)
y <- c(
  rpois(300, exp(x[1:300, ] %*% theta_0)),
  rpois(400, exp(x[301:700, ] %*% (theta_0 + delta))),
  rpois(300, exp(x[701:1000, ] %*% theta_0)),
  rpois(100, exp(x[1001:1100, ] %*% (theta_0 - delta))),
  rpois(200, exp(x[1101:1300, ] %*% theta_0)),
  rpois(200, exp(x[1301:1500, ] %*% (theta_0 + delta)))
)
result <- fastcpd(
  formula = y ~ . - 1,
  data = data.frame(y = y, x = x),
  beta = (p + 1) * log(1500) / 2,
  k = function(x) 0,
  family = "poisson",
  epsilon = 1e-5
)
summary(result)

# Penalized linear regression
library(fastcpd)
```

```
set.seed(1)
n <- 1500
p_true <- 6
p <- 50
x <- mvtnorm::rmvnorm(1500, rep(0, p), diag(p))
theta_0 <- rbind(
  runif(p_true, -5, -2),
  runif(p_true, -3, 3),
  runif(p_true, 2, 5),
  runif(p_true, -5, 5)
)
theta_0 <- cbind(theta_0, matrix(0, ncol = p - p_true, nrow = 4))
y <- c(
  x[1:300, ] %*% theta_0[1, ] + rnorm(300, 0, 1),
  x[301:700, ] %*% theta_0[2, ] + rnorm(400, 0, 1),
  x[701:1000, ] %*% theta_0[3, ] + rnorm(300, 0, 1),
  x[1001:1500, ] %*% theta_0[4, ] + rnorm(500, 0, 1)
)
result <- fastcpd(
  formula = y ~ . - 1,
  data = data.frame(y = y, x = x),
  family = "lasso"
)
plot(result)
summary(result)

# Custom cost function: logistic regression
library(fastcpd)
set.seed(1)
p <- 5
x <- matrix(rnorm(375 * p, 0, 1), ncol = p)
theta <- rbind(rnorm(p, 0, 1), rnorm(p, 2, 1))
y <- c(
  rbinom(200, 1, 1 / (1 + exp(-x[1:200, ] %*% theta[1, ]))),
  rbinom(175, 1, 1 / (1 + exp(-x[201:375, ] %*% theta[2, ])))
)
data <- data.frame(y = y, x = x)
result_builtin <- fastcpd(
  formula = y ~ . - 1,
  data = data,
  family = "binomial"
)
logistic_loss <- function(data, theta) {
  x <- data[, -1]
  y <- data[, 1]
  u <- x %*% theta
  nll <- -y * u + log(1 + exp(u))
  nll[u > 10] <- -y[u > 10] * u[u > 10] + u[u > 10]
  sum(nll)
}
logistic_loss_gradient <- function(data, theta) {
  x <- data[nrow(data), -1]
  y <- data[nrow(data), 1]
  c(-(y - 1 / (1 + exp(-x %*% theta)))) * x
}
logistic_loss_hessian <- function(data, theta) {
  x <- data[nrow(data), -1]
```

```
  prob <- 1 / (1 + exp(-x %*% theta))
  (x %o% x) * c((1 - prob) * prob)
}
result_custom <- fastcpd(
  formula = y ~ . - 1,
  data = data,
  epsilon = 1e-5,
  cost = logistic_loss,
  cost_gradient = logistic_loss_gradient,
  cost_hessian = logistic_loss_hessian
)
cat(
  "Change points detected by built-in logistic regression model: ",
  result_builtin@cp_set, "\n",
  "Change points detected by custom logistic regression model: ",
  result_custom@cp_set, "\n",
  sep = ""
)

# Custom cost function: mean shift
library(fastcpd)
set.seed(1)
p <- 1
data <- rbind(
  mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(100, p)),
  mvtnorm::rmvnorm(400, mean = rep(50, p), sigma = diag(100, p)),
  mvtnorm::rmvnorm(300, mean = rep(2, p), sigma = diag(100, p))
)
segment_count_guess <- 10
block_size <- max(floor(sqrt(nrow(data)) / (segment_count_guess + 1)), 2)
block_count <- floor(nrow(data) / block_size)
data_all_vars <- rep(0, block_count)
for (block_index in seq_len(block_count)) {
  block_start <- (block_index - 1) * block_size + 1
  block_end <- if (block_index < block_count) block_index * block_size else nrow(data)
  data_all_vars[block_index] <- var(data[block_start:block_end, ])
}
data_all_var <- mean(data_all_vars)
mean_loss <- function(data) {
  n <- nrow(data)
  (norm(data, type = "F")^2 - colSums(data)^2 / n) / 2 / data_all_var +
    n / 2 * (log(data_all_var) + log(2 * pi))
}
mean_loss_result <- fastcpd(
  formula = ~ . - 1,
  data = data.frame(data),
  beta = (p + 1) * log(nrow(data)) / 2,
  p = p,
  cost = mean_loss
)
summary(mean_loss_result)

# Custom cost function: variance change
library(fastcpd)
set.seed(1)
p <- 1
data <- rbind.data.frame(
```

```
  mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(1, p)),
  mvtnorm::rmvnorm(400, mean = rep(0, p), sigma = diag(50, p)),
  mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(2, p))
)
data_all_mu <- colMeans(data)
var_loss <- function(data) {
  demeaned_data_norm <- norm(sweep(data, 2, data_all_mu), type = "F")
  nrow(data) * (1 + log(2 * pi) + log(demeaned_data_norm^2 / nrow(data))) / 2
}
var_loss_result <- fastcpd(
  formula = ~ . - 1,
  data = data,
  beta = (p + 1) * log(nrow(data)) / 2,
  p = p,
  cost = var_loss
)
summary(var_loss_result)

# Custom cost function: mean shift and variance change
library(fastcpd)
set.seed(1)
p <- 1
data <- rbind.data.frame(
  mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(1, p)),
  mvtnorm::rmvnorm(400, mean = rep(10, p), sigma = diag(1, p)),
  mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(50, p)),
  mvtnorm::rmvnorm(300, mean = rep(0, p), sigma = diag(1, p)),
  mvtnorm::rmvnorm(400, mean = rep(10, p), sigma = diag(1, p)),
  mvtnorm::rmvnorm(300, mean = rep(10, p), sigma = diag(50, p))
)
meanvar_loss <- function(data) {
  loss_part <- (colSums(data^2) - colSums(data)^2 / nrow(data)) / nrow(data)
  nrow(data) * (1 + log(2 * pi) + log(loss_part)) / 2
}
meanvar_loss_result <- fastcpd(
  formula = ~ . - 1,
  data = data,
  beta = (2 * p + 1) * log(nrow(data)) / 2,
  p = 2 * p,
  cost = meanvar_loss
)
summary(meanvar_loss_result)

# Custom cost function: Huber loss
library(fastcpd)
set.seed(1)
n <- 400 + 300 + 500
p <- 5
x <- mvtnorm::rmvnorm(n, mean = rep(0, p), sigma = diag(p))
theta <- rbind(
  mvtnorm::rmvnorm(1, mean = rep(0, p - 3), sigma = diag(p - 3)),
  mvtnorm::rmvnorm(1, mean = rep(5, p - 3), sigma = diag(p - 3)),
  mvtnorm::rmvnorm(1, mean = rep(9, p - 3), sigma = diag(p - 3))
)
theta <- cbind(theta, matrix(0, 3, 3))
theta <- theta[rep(seq_len(3), c(400, 300, 500)), ]
y_true <- rowSums(x * theta)
```

```
factor <- c(
  2 * stats::rbinom(400, size = 1, prob = 0.95) - 1,
  2 * stats::rbinom(300, size = 1, prob = 0.95) - 1,
  2 * stats::rbinom(500, size = 1, prob = 0.95) - 1
)
y <- factor * y_true + stats::rnorm(n)
data <- cbind.data.frame(y, x)
huber_threshold <- 1
huber_loss <- function(data, theta) {
  residual <- data[, 1] - data[, -1, drop = FALSE] %*% theta
  indicator <- abs(residual) <= huber_threshold
  sum(
    residual^2 / 2 * indicator +
      huber_threshold * (abs(residual) - huber_threshold / 2) * (1 - indicator)
  )
}
huber_loss_gradient <- function(data, theta) {
  residual <- c(data[nrow(data), 1] - data[nrow(data), -1] %*% theta)
  if (abs(residual) <= huber_threshold) {
    -residual * data[nrow(data), -1]
  } else {
    -huber_threshold * sign(residual) * data[nrow(data), -1]
  }
}
huber_loss_hessian <- function(data, theta) {
  residual <- c(data[nrow(data), 1] - data[nrow(data), -1] %*% theta)
  if (abs(residual) <= huber_threshold) {
    outer(data[nrow(data), -1], data[nrow(data), -1])
  } else {
    0.01 * diag(length(theta))
  }
}
huber_regression_result <- fastcpd(
  formula = y ~ . - 1,
  data = data,
  beta = (p + 1) * log(n) / 2,
  cost = huber_loss,
  cost_gradient = huber_loss_gradient,
  cost_hessian = huber_loss_hessian
)
summary(huber_regression_result)
```

`fastcpd-class` *An S4 class to store the output created with fastcpd*

Description

This S4 class stores the output from fastcpd. A fastcpd object consist of several slots including the call to fastcpd, the data used, the family of the model, the change points, the cost values, the residuals, the estimated parameters and a boolean indicating whether the model was fitted with only change points or with change points and parameters, which you can select using `@`.

Slots

`call` The call to fastcpd.

`data` The data used.

`family` The family of the model.

`cp_set` The change points.

`cost_values` The cost values for each segment.

`residuals` The residuals for each segment.

`thetas` The estimated parameters for each segment.

`cp_only` A boolean indicating whether the model was fitted with only change points or with change points and parameters.

`plot.fastcpd` *Plot the data and the change points for a* `fastcpd` *object*

Description

Plot the data and the change points for a `fastcpd` object

Usage

```
## S3 method for class 'fastcpd'
plot(x, ...)

## S4 method for signature 'fastcpd,missing'
plot(x, y, ...)
```

Arguments

`x`	`fastcpd` object.
`...`	Ignored.
`y`	Ignored.

`print.fastcpd` *Print the call and the change points for a* `fastcpd` *object*

Description

Print the call and the change points for a `fastcpd` object

Usage

```
## S3 method for class 'fastcpd'
print(x, ...)

## S4 method for signature 'fastcpd'
print(x, ...)
```

Arguments

`x`	`fastcpd` object.
`...`	Ignored.

show.fastcpd — *Show the available methods for a* `fastcpd` *object*

Description

Show the available methods for a `fastcpd` object

Usage

```
## S3 method for class 'fastcpd'
show(object)

## S4 method for signature 'fastcpd'
show(object)
```

Arguments

`object`	`fastcpd` object.

summary.fastcpd — *Show the summary of a* `fastcpd` *object*

Description

Show the summary of a `fastcpd` object

Usage

```
## S3 method for class 'fastcpd'
summary(object, ...)

## S4 method for signature 'fastcpd'
summary(object, ...)
```

Arguments

`object`	`fastcpd` object.
`...`	Ignored.

Index