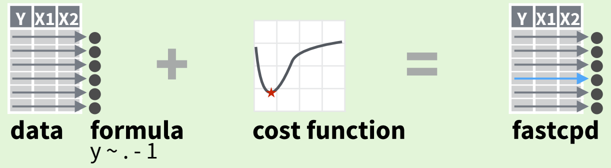
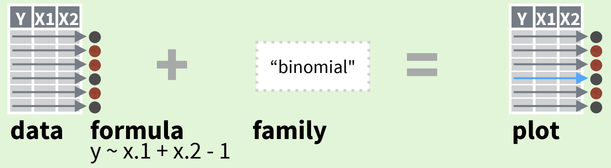
Change point analysis with fastcpd : : **CHEATSHEET**

Basics

fastcpd is based on the **Sequential Gradient Descent** and **Penalized Exact Linear Time**, avoiding repeated likelihood calculation and pruning impossible change points given a **data** set, a **cost** function, and optional **gradient / Hessian**.



Various built-in families are provided to better utilize the improved performance. A model of as simple as **data** + **family** is enough.



Complete the template below to find change points.

fastcpd (data = <DATA>,
[family = "<FAMILY>" OR cost = <COST_FUNCTION>], **required**
formula = **<FORMULA>**, beta = **<NUMERIC>**,
segment_count = **<INTEGER>**, p = **<INTEGER>**,
trim = **<NUMERIC>**, k = **<FUNCTION(x)>**,
cost_gradient = **<FUNCTION(data, theta)>**,
cost_hessian = **<FUNCTION(data, theta)>**, …) **Not required, sensible defaults supplied**

fastcpd(data = data, …**)** Returns a “fastcpd" object containing the information used to call the method.

plot(fastcpd_result**)** Invokes `ggplot2` to plot the data.

summary(fastcpd_result**)** Outputs summary information of the call, including change point locations, estimated parameters and residuals for each segments.

family

Built-in families ready to use.

Regression and **time series** (case sensitive)

Regression - "gaussian", "binomial", "poisson", "lasso"

Time series - "ar", "var"

fastcpd

Use fastcpd to deal with all data types including the built-in families and any custom models, and fastcpd_ts to deal with ar(p) and var(p) time series.

LINEAR REGRESSION

```
result <– fastcpd(y ~ x.1 + x.2 - 1,
  data = data.frame(y = y, x = x), family = "gaussian", …)
```

… = **segment_count** = 10, **trim** = 0.05

… = **beta** = (p + 1) * log(nrow(data)) / 2 * variance

… = **mementum_coef** = 0.1, **cp_only** = TRUE

```
… = k = function(x) {
      if (x < n / 4) 2
      else if (x < n / 2) 1
      else 0
    }
```

… = **vanilla_percentage** = 0, **p** = ncol(data) - 1

… = all possible combinations from above

epsilon: ignored **min_prob**: ignored

winsorise_minval/winsorise_maxval: ignored

cost/cost_gradient/cost_hessian: incompatible

PENALIZED LINEAR REGRESSION

```
result <– fastcpd(y ~ . - 1,
  data = data.frame(y = y, x = x), family = "lasso", …)
```

… = **segment_count** = 10, **trim** = 0.025

… = **beta** = (p + 1) * log(nrow(data)) / 2

… = **mementum_coef** = 0, **cp_only** = TRUE

```
… = k = function(x) {
      if (x < n / 4) 1
      else 0
    }
```

… = **vanilla_percentage** = 0, **p** = ncol(data) - 1

… = all possible combinations from above

epsilon: ignored **min_prob**: ignored

winsorise_minval/**winsorise_maxval**: ignored

cost/cost_gradient/cost_hessian: incompatible

LOGISTIC REGRESSION

```
result <– fastcpd(y ~ . - 1,
  data = data.frame(y = y, x = x), family = "binomial", …)
```

… = **segment_count** = 8, **cp_only** = FALSE

… = **beta** = (p + 1) * log(nrow(data)) / 2

… = **mementum_coef** = 0, **trim** = 0.03

```
… = k = function(x) {
      if (x < n / 4) 1
      else 0
    }
```

… = **vanilla_percentage** = 0.1, **epsilon** = 1e-10

… = **p** = ncol(data) - 1

… = all possible combinations from above

min_prob: ignored

winsorise_minval/winsorise_maxval: ignored

cost/cost_gradient/cost_hessian: incompatible

AR(p)

```
result <– fastcpd.ts(x, "ar", 3, …)
result <– fastcpd( ~ x - 1, data.frame(x), family = "ar", p = 3, …)
```

… = **segment_count** = 8, **cp_only** = FALSE

… = **beta** = (p + 1) * log(nrow(data)) / 2 * variance

… = **mementum_coef** = 0, **trim** = 0.03

```
… = k = function(x) {
      if (x < n / 4) 1
      else 0
    }
```

… = **vanilla_percentage** = 0.1

… = all possible combinations from above

epsilon: ignored **min_prob**: ignored

winsorise_minval/winsorise_maxval: ignored

cost/cost_gradient/cost_hessian: incompatible

POISSON REGRESSION

```
result <– fastcpd(y ~ . - 1,
  data = data.frame(y = y, x = x), family = "poisson", …)
```

… = **segment_count** = 6, **cp_only** = TRUE

… = **beta** = (p + 1) * log(nrow(data)) / 2

… = **mementum_coef** = 0.02, **trim** = 0.03

```
… = k = function(x) {
      if (x < n / 4) 1
      else 0
    }
```

… = **vanilla_percentage** = 0.15, **epsilon** = 1e-5

… = **p** = ncol(data) - 1, **min_prob** = 10^10

… = **winsorise_minval** = -20

… = **winsorise_maxval** = 20

… = all possible combinations from above

cost/cost_gradient/cost_hessian: incompatible

VAR(p)

```
result <– fastcpd_ts(x, "var", 3, …)
result <– fastcpd( ~ . - 1, data.frame(x), family = "var", p = 3, …)
```

… = **segment_count** = 6, **cp_only** = TRUE

… = **beta** = (p + 1) * log(nrow(data)) / 2

… = **mementum_coef** = 0.02, **trim** = 0.03

```
… = k = function(x) {
      if (x < n / 4) 1
      else 0
    }
```

… = **vanilla_percentage** = 0.15

… = all possible combinations from above

epsilon: ignored **min_prob**: ignored

winsorise_minval/winsorise_maxval: ignored

cost/cost_gradient/cost_hessian: incompatible

UTILITY FUNCTION: PRINT

```
r$> print(result_binomial)

Change points:
[1] 126
```

```
r$> print(result_no_cp)

No change points found
```

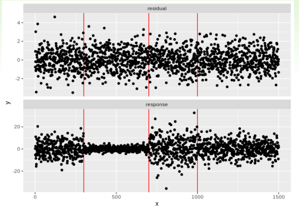
```
r$> print(result_custom)

Change points:
[1] 300 700
```

UTILITY FUNCTION: SUMMARY

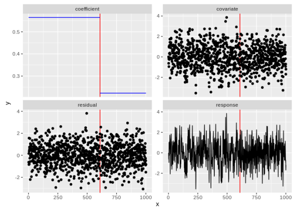
```
r$> summary(result_gaussian)

Call:
fastcpd(y ~ . - 1, data = data, family = "gaussian")

Change points:
98 202

Cost values:
53.44023 53.1441 45.04974

Parameters:
  segment 1   segment 2  segment 3
1  0.9704022 -1.07884004  0.5925092
2  1.1786074 -0.01757927 -0.5287126
```

UTILITY FUNCTION: PLOT

r$> plot(result_lasso)



r$> plot(result_ar1)



Authors: Xingchi LI (xingchi.li) • Xianyang Zhang (zhangxiany-tamu.github.io) • Trisha Dawn (trisha@stat.tamu.edu) • fastcpd.xingchi.li • fastcpd 0.8.1 • Updated: 2023-10