tytul

autor

data

Algorytm jest dostępny w specjalnie przygotowanym pakiecie o nazwie `coxphSGD` napisanym w języku $\mathcal{R}$, który można pobrać z internetu i zainstalować poleceniem

```
if (packageVersion("devtools") < 1.6) {
  install.packages("devtools")
}
devtools::install_github("MarcinKosinski/Cox-SGD")
```

Poniżej przedstawione są argumenty, które przyjmuje funkcja `coxphSGD()`, która estymuje współczynniki w modelu proporcjonalnych hazardów Coxa metodą stochastycznego spadku gradientu. Starano zachować jednorodność kolejności i nazewnictwa parametrów z funkcją `coxph` z pakietu `survival` [**?**], [**?**].

```
#' Stochastic Gradient Descent log-likelihood estimation in
#' Cox proportional hazards model
#'
#' Function \code{coxphSGD} estimates coefficients using stochastic
#' gradient descent algorithm in Cox proportional hazards model.
#'
#' @param formula a formula object, with the response on the left of a ~ operator,
#' and the terms on the right. The response must be a survival object as returned by
#' the Surv function.
#' @param data a data.frame in which to interpret the variables named in the \code{formula}.
#' @param reorderObs a logical value telling whether reorder observations at each epoch.
#' when order of observations in estimation should be randomly generated.
#' @param learningRates a function specifing how to define learning rates in
#' steps of the algorithm. By default the \code{f(t)=1/t} is used, where \code{t} is
#' the number of algorithm's step.
#' @param beta_0 a numeric vector (if of length 1 then will be replicated) of length
#' equal to the number of variables after using \code{formula} in the \code{model.matrix}
#' function
#' @param epsilon a numeric value with the stop condition of the estimation algorithm.
#' @param epoch a numeric value declaring the number of epoches to run for the
#' estimation algorithm in the stochastic gradient descent.
#' @param batchSize a numeric value specifing the size of a batch set to take from
#' the reordered dataset to update the coefficients in one step of an algorithm.
#'
#' @note If one of the conditions is fullfiled
#' \itemize{
#'  \item \eqn{||\beta_{j+1}-\beta_{j}|| <}\code{epsilon} parameter for any \eqn{j}
#'  \item \eqn{#epochs >} \code{epochs} parameter
#' }
#' the estimation process is stopped.
#' @export
#' @importFrom survival Surv
#' @importFrom assertthat assert_that
#' @examples
```

```
#' library(survival)
#' \dontrun{
#' coxphSGD(Surv(time, status) ~ ph.ecog + age, data=lung)
#' }
#'
```

Sprawdzenie parametrów na wejściu funkcji.

```
coxphSGD = function(formula, data, reorderObs = TRUE,
                    learningRates = function(x) 1/x,
                    beta_0 = 0, epsilon = 1e-5,
                    batchSize = 10, epoch = 20 ) {

  assert_that(is.data.frame(data))
  assert_that(is.logical(reorderObs))
  assert_that(is.function(learningRates))
  assert_that(is.numeric(epsilon))
  assert_that(is.numeric(epoch) & epoch > 0)
```

Identyfikacja przekazanych parametrów. Poniższa identyfikacja bazuje na kodzie funkcji `coxph()`.

```
  Call <- match.call()
  indx <- match(c("formula", "data", "order", "learningRates",
                  "epsilon", "batchsize", "epoch"),
                names(Call), nomatch = 0)
  if (indx[1] == 0)
      stop("A formula argument is required")
  temp <- Call[c(1, indx)]
  temp[[1]] <- as.name("model.frame")

  mf <- eval(temp, parent.frame())
  Y <- model.extract(mf, "response")

  if (!inherits(Y, "Surv"))
      stop("Response must be a survival object")
  type <- attr(Y, "type")

  if (type != "right" && type != "counting")
      stop(paste("Cox model doesn't support \"", type, "\" survival data",
          sep = ""))
  if (length(beta_0) == 1) {
    beta_0 <- rep(beta_0, ncol(mf)-1)
  }
```

Początkowa zamiana kolejności obserwacji w wejściowym zbiorze.

```
  if (reorderObs) {
    obsOrder <- sample(1:nrow(data))
    mf <- mf[obsOrder, ]
    Y <- Y[obsOrder, ]
  }
```

Wprowadzenie zmiennych pomocniczych.

```
j <- 0 # number of an algorithm's step
diff <- 0 # differences between estimates along steps
i <- 0 # indicator of a batch sample
n <- nrow(data)
batchSamplesStarts <- seq(1,n, batchSize) # indexes of starts of batch samples
epochs_n <- 1# indicator of the present epochs number
beta_j <- beta_0
```

gdzie

$$\text{diff}_j = \| \beta_{j+1} - \beta_j \| < \epsilon.$$

Sprawdzenie warunku zbieżności algorytmu.

```
while ( j == 0 | (diff < eps & epochs_n <= epoch) ){
  j <- j+1
  i <- i+1
```

Rozpoczęcie algorytmu. Dla losowej kolejności obserwacji, weź pierwszą porcję obserwacji, której wielkość ustawiona jest dzięki parametrowi `batchSize`. Tak powstaje podzbiór obserwacji oznaczany przez $\mathcal{B}$, dla którego zachodzi $|\mathcal{B}| = b$, a b odpowiada wartości ustawionej w parametrze `batchSize`. Indeksy obserwacji należace do zbioru $\mathcal{B}$ zdefiniujmy jako $\mathcal{B}_{\text{ind}} = \{i : X_i \in \mathcal{B}\}$.

```
if (i < length(batchSamplesStarts)-1){
  batchSample_variables <- mf[batchSamplesStarts[i]:(batchSamplesStarts[i]+batchSize-1), ]
  batchSample_response <- Y[batchSamplesStarts[i]:(batchSamplesStarts[i]+batchSize-1), ]
} else {
  if (i == length(batchSamplesStarts)-1) {
    # last batch sample can me shorter than all others
    batchSample_variables <- mf[batchSamplesStarts[i]:(n), ]
    batchSample_response <- Y[batchSamplesStarts[i]:(n), ]
  } else {
    i <- 1
    batchSample_variables <- mf[batchSamplesStarts[i]:(batchSamplesStarts[i]+batchSize-1), ]
    batchSample_response <- Y[batchSamplesStarts[i]:(batchSamplesStarts[i]+batchSize-1), ]
    epochs_n <- epochs_n + 1 # epoch has passed
    # so reorder samples
      if (reorderObs) {
        obsOrder <- sample(1:nrow(data))
        mf <- mf[obsOrder, ]
        Y <- Y[obsOrder, ]
      }
  }
}
```

Dla danego podzbioru wyznacz odpowiadającą jemu część pochodnej częściowej funkcji log-wiarogoności ze zmienionym znakiem. Ponieważ omawiane algorytmy rozwiązuja problem minimalizacji badanej funkcji, zaś celem estymacji w modelu Coxa jest znalezienie parametrów modelu maksymalizujących funkcję częściowej log-wiarogodności, zatem wzięcie do minimalizacji funkcji z przeciwnym znakiem doprowadzi do wykorzystania metod znajdujących minimum do znalezienia maksimum. Dla j-ego kroku algorytmu i k-tej pochodnej cząstkowej dysponuje się podzbiorem $\mathcal{B}$ o liczności b (parametr `batchSize`), wtedy

$$-U_k^{\mathcal{B}}(\beta_j) = -\sum_{i \in \mathcal{B}_{\text{ind}}} U_{k_i}^{\mathcal{B}}(\beta_j) = -\sum_{i \in \mathcal{B}_{\text{ind}}} Y_i \Big(X_{ik} - \frac{\sum\limits_{l \in \mathscr{R}_{\mathcal{B}}(t_i)} X_{lk} e^{X_l' \beta_j}}{\sum\limits_{l \in \mathscr{R}_{\mathcal{B}}(t_i)} e^{X_l' \beta_j}} \Big),$$

gdzie b oznacza wielkość podzbioru $\mathcal{B}$, zaś $\mathscr{R}_{\mathcal{B}}(t_i)$ to zbiór ryzyka dla podzbioru $\mathcal{B}$ w czasie t_i.

```
  U_ik <- matrix(0, ncol = ncol(mf)-1,
                 nrow = nrow(batchSample_variables))
  U_k <- numeric(ncol(mf)-1)
  for ( k in 2:ncol(mf)) { # 1st dimension is Y
    for (i in 1:nrow(batchSample_variables)){
      l <- which(batchSample_response[, 1] <= batchSample_response[i, 1])
      U_ik[i,k] <- -batchSample_variables[i, k] + sum(batchSample_variables[l, k]*
                                          exp(batchSample_variables[l, ]%*%beta_j)/
            sum(exp(batchSample_variables[l, ]%*%beta_j))

    }
  U_k[k] <- sum(U_ik[, k])
  }
```

Następnie zaktualizuj parametry modelu.

$$\beta_{k_{j+1}} = \beta_{k_j} - \alpha_j U_k^{\mathcal{B}}(\beta_{k_j})$$

```
   beta_j <- beta_j - learningRates(j)*U_k
```

Przypisz nowy warunek stopu.

```
   diff <- sqrt(sum(learningRates(j)*U_k))
  }
```

Zwrócenie parametrów funkcji, gdy spełniony chociaż jeden warunek stopu.

```
  fit <- list()
  fit$Call <- Call
  fit$mf <- mf
  fit$coeff <- beta_j
  fit$epochs_n <- epochs_n
  fit
}
#coxphSGD(Surv(time, status) ~ ph.ecog + age, data=lung)
```