Untitled

Marcin Kosinski

25.10.2015

```
logitGD <- function(y, x, optim.method = "GDI", eps = 10e-4,
                    max.iter = 100, alpha = function(t){1/t}, beta_0 = c(0,0)){
  stopifnot(length(y) == length(x) & optim.method %in% c("GDI", "GDII", "SGDI")
            & is.numeric(c(max.iter, eps, x)) & all(c(eps, max.iter) > 0) &
              is.function(alpha))
  iter <- 0
  err <- list()
  err[[iter+1]] <- eps+1
  w_old <- beta_0

  res <-list()
  while(iter < max.iter && (abs(err[[ifelse(iter==0,1,iter)]]) > eps)){

    iter <- iter + 1
    if (optim.method == "GDI"){
      w_new <- w_old + alpha(iter)*updateWeightsGDI(y, x, w_old)
    }
    if (optim.method == "GDII"){
      w_new <- w_old + as.vector(inverseHessianGDII(x, w_old)%*%
                                   updateWeightsGDI(y, x, w_old))
    }
    if (optim.method == "SGDI"){
      w_new <- w_old + alpha(iter)*updateWeightsSGDI(y[iter], x[iter], w_old)
    }
    res[[iter]] <- w_new
    err[[iter]] <- sqrt(sum((w_new - w_old)^2))

    w_old <- w_new

  }
  return(list(steps = c(list(beta_0),res), errors = c(list(c(0,0)),err)))
}

updateWeightsGDI <- function(y, x, w_old){
  (1/length(y))*c(sum(y-p(w_old, x)), sum(x*(y-p(w_old, x))))
  #c(sum(y-p(w_old, x)), sum(x*(y-p(w_old, x))))
}

updateWeightsSGDI <- function(y_i, x_i, w_old){
  c(y_i-p(w_old, x_i), x_i*(y_i-p(w_old, x_i)))
}

p <- function(w_old, x_i){
  1/(1+exp(-w_old[1]-w_old[2]*x_i))
}

inverseHessianGDII <- function(x, w_old){
```

```
  solve(
    matrix(c(
      sum(p(w_old, x)*(1-p(w_old, x))),
      sum(x*p(w_old, x)*(1-p(w_old, x))),
      sum(x*p(w_old, x)*(1-p(w_old, x))),
      sum(x*x*p(w_old, x)*(1-p(w_old, x)))
    ),
    nrow =2 )
  )
}
```

```
                    # wstępna inicjalizacja parametrów
eps = 1e-5                              # warunek stopu.

n = length(data)                        # data jest listą ramek danych.

diff = eps + 1                          # różnice w oszacowaniach parametrów
                                        # między kolejnymi krokami.

learningRates = function(x) 1/x         # długości kroku algorytmu.

beta_old = numeric(0, length = k)       # punkt startowy dlugosci k,
                                        # gdzie k to liczba zmiennych
                                        # objaśniających w modelu.

max.iter = 500                          # maksymalna liczba kroków.
ź
                               # estymacja
i = 1                                   # iterator kroku algorytmu.
while(i <= max.iter | diff < eps) do
  iter = ifelse(i mod n == 0, n, i mod n)# wybierz kolejny podzbiór batch.
  batch = data[[iter]]
  beta_new = beta_old - learningRates(i) * U_Batch(batch)
                                        # U_Batch to częściowa funkcja
                                        # log-wiarogdności dla zaobserwowanego
                                        # zbioru `batch`
  diff = euclidean_dist(beta_new, beta_old) # odległość euklidesowa
  beta_old = beta_new
  i = i + 1
end while
return beta_new
```

```
coxphSGD <- function(formula, data, learningRates = function(x){1/x},
                     beta_0 = 0, epsilon = 1e-5, max.iter = 500 ) {
  checkArguments(formula, data, learningRates,
                 beta_0, epsilon) -> beta_start # check arguments
  n <- length(data)
  diff <- epsilon + 1
  i <- 1
  beta_new <- list()     # steps are saved in a list so that they can
  beta_old <- beta_start # be tracked in the future
  # estimate
  while(i <= max.iter & diff > epsilon) {
```

```
    beta_new[[i]] <- coxphSGD_batch(formula = formula, beta = beta_old,
        learningRate = learningRates(i), data = data[[ifelse(i%%n==0,n,i%%n)]])

    diff <- sqrt(sum((beta_new[[i]] - beta_old)^2))
    beta_old <- beta_new[[i]]
    i <- i + 1
  }
  # return results
  list(Call = match.call(), epsilon = epsilon, learningRates = learningRates,
       steps = i, coefficients = c(list(beta_start), beta_new))
}

coxphSGD_batch <- function(formula, data, learningRate, beta){
  # collect times, status, variables and reorder samples
  # to make the algorithm more clear to read and track
  batchData <- prepareBatch(formula = formula, data = data)

  # calculate the log-likelihood for this batch sample
  partial_sum <- list()

  for(k in 1:nrow(batchData)) {
    # risk set for current time/observation
    risk_set <- batchData %>% filter(times >= batchData$times[k])

    nominator <- apply(risk_set[, -c(1,2)], MARGIN = 1, function(element){
      element * exp(element * beta)
    }) %>% rowSums()

    denominator <- apply(risk_set[, -c(1,2)], MARGIN = 1, function(element){
      exp(element * beta)
    }) %>% rowSums()

    partial_sum[[k]] <-
      batchData[k, "event"] * (batchData[k, -c(1,2)] - nominator/denominator)
  }
  do.call(rbind, partial_sum) %>%
    colSums() -> U_batch

  return(beta + learningRate * U_batch)
}

checkArguments <- function(formula, data, learningRates,
                           beta_0, epsilon) {
  assert_that(is.list(data) & length(data) > 0)
  assert_that(length(unique(unlist(lapply(data, ncol)))) == 1)
  # + check names and types for every variables
  assert_that(is.function(learningRates))
  assert_that(is.numeric(epsilon))
  assert_that(is.numeric(beta_0))

    # check length of the start parameter
  if (length(beta_0) == 1) {
    beta_0 <- rep(beta_0, as.character(formula)[3] %>%
```

```
                    strsplit("\\+") %>%
                    unlist %>%
                    length)
  }
  return(beta_0)
}
```

```
x <- runif(1000)
z <- 2 + 3*x
pr <- 1/(1+exp(-z))
y <- rbinom(1000,1,pr)


logitGD(y, x, optim.method = "GDI", eps = 10e-5, max.iter = 500)$steps -> GDI
logitGD(y, x, optim.method = "GDII", eps = 10e-5, max.iter = 500)$steps -> GDII

ind <- sample(length(y))
logitGD(y[ind], x[ind], optim.method = "SGDI",
        max.iter = 500, eps = 10e-5)$steps -> SGDI.1
ind2 <- sample(length(y))
logitGD(y[ind2], x[ind2], optim.method = "SGDI",
        max.iter = 500, eps = 10e-5)$steps -> SGDI.2
ind3 <- sample(length(y))
logitGD(y[ind3], x[ind3], optim.method = "SGDI",
        max.iter = 500, eps = 10e-5)$steps -> SGDI.3
ind4 <- sample(length(y))
logitGD(y[ind4], x[ind4], optim.method = "SGDI",
        max.iter = 500, eps = 10e-5)$steps -> SGDI.4
ind5 <- sample(length(y))
logitGD(y[ind5], x[ind5], optim.method = "SGDI",
        max.iter = 500, eps = 10e-5)$steps -> SGDI.5

do.call(rbind, c(GDI, GDII, SGDI.1, SGDI.2, SGDI.3, SGDI.4, SGDI.5)) -> coeffs
unlist(lapply(list(GDI, GDII, SGDI.1, SGDI.2, SGDI.3, SGDI.4, SGDI.5), length)) -> algorithm
data2viz <- cbind(as.data.frame(coeffs),
      algorithm = unlist(mapply(rep, c("GDI", "GDII", "SGDI.1", "SGDI.2", "SGDI.3", "SGDI.4", "SGDI.5")
names(data2viz)[1:2] <- c("Intercept", "X")
library(ggplot2); library(ggthemes)
ggplot(data2viz) +
  geom_point(aes(x = X, y = Intercept, col = algorithm)) +
  geom_line(aes(x = X, y = Intercept, col = algorithm,
                group = algorithm)) +
  theme_tufte(base_size = 20)
```

logitGD() asda graphSGD()

```
graphSGD(c(0,0), y, x)
graphSGD(c(3.1,2.1), y, x)
graphSGD(c(4,3), y, x)
graphSGD(c(1,2), y, x)
```

```r
dataCox <- function(N, lambda, rho, x, beta, censRate){

  # real Weibull times
  u <- runif(N)
  Treal <- (- log(u) / (lambda * exp(x %*% beta)))^(1 / rho)

  # censoring times
  Censoring <- rexp(N, censRate)

  # follow-up times and event indicators
  time <- pmin(Treal, Censoring)
  status <- as.numeric(Treal <= Censoring)

  # data set
  data.frame(id=1:N, time=time, status=status, x=x)
}

x <- matrix(sample(0:1, size = 40, replace = TRUE), ncol = 2)

head(dataCox(20, 3, 2, x, beta = c(2,3), 5))
```

```
  id       time status x.1 x.2
1  1 0.07591466      1   0   1
2  2 0.21533677      1   0   1
3  3 0.09686408      0   1   0
4  4 0.01157519      0   1   1
5  5 0.07779189      0   0   1
6  6 0.11807808      0   1   0
```

```r
graphSGD(c(0,0), y, x, 4561);graphSGD(c(0,0), y, x, 456)
graphSGD(c(2,1), y, x, 4561);graphSGD(c(2,1), y, x, 456);
graphSGD(c(1,0), y, x, 4561);graphSGD(c(1,0), y, x, 456);
graphSGD(c(2.1,3.1), y, x, 4561)graphSGD(c(2.1,3.1), y, x, 456)
```

```r
x <- matrix(sample(0:1, size = 20000, replace = TRUE), ncol = 2)
dCox <- dataCox(10^4, lambda = 3, rho = 2, x, beta = c(1,3), censRate = 5)
vizCoxSGD(dCox)
```

```
coxphSGD <- function(formula, data, learningRates = function(x){1/x},
                     beta_0 = 0, epsilon = 1e-5, max.iter = 500 ) {
  checkArguments(formula, data, learningRates,
                 beta_0, epsilon) -> beta_start # check arguments
  n <- length(data)
  diff <- epsilon + 1
  i <- 1
  beta_new <- list()     # steps are saved in a list so that they can
  beta_old <- beta_start # be traced in the future
  # estimate
  while(i <= max.iter & diff > epsilon) {
    beta_new[[i]] <- coxphSGD_batch(formula = formula, beta = beta_old,
        learningRate = learningRates(i), data = data[[ifelse(i%%n==0,n,i%%n)]]) %>%
      unlist
    diff <- sqrt(sum((beta_new[[i]] - beta_old)^2))
    beta_old <- beta_new[[i]]
    i <- i + 1  ; cat("\r iteration: ", i, "\r")
  }  # return results
  list(Call = match.call(), epsilon = epsilon, learningRates = learningRates,
       steps = i, coefficients = c(list(beta_start), beta_new))
}

coxphSGD_batch <- function(formula, data, learningRate, beta){
  # collect times, status, variables and reorder samples
  # to make the algorithm more clear to read and track
  batchData <- prepareBatch(formula = formula, data = data)
  # calculate the log-likelihood for this batch sample
  partial_sum <- list()
  foreach(k = 1:nrow(batchData)) %do% {
    # risk set for current time/observation
    risk_set <- batchData %>% filter(times >= batchData$times[k])

    nominator <- apply(risk_set[, -c(1,2)], MARGIN = 1, function(element){
      element * exp(element * beta)
    }) %>% rowSums()

    denominator <- apply(risk_set[, -c(1,2)], MARGIN = 1, function(element){
      exp(element * beta)
    }) %>% rowSums()

    partial_sum[[k]] <-
      batchData[k, "event"] * (batchData[k, -c(1,2)] - nominator/denominator)
  }
  do.call(rbind, partial_sum) %>%
    colSums() -> U_batch

  return(beta + learningRate * U_batch)
}
```

```
prepareBatch <- function(formula, data) {
  # Parameter identification as in  `survival::coxph()`.
  Call <- match.call()
  indx <- match(c("formula", "data"),
```

```
                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 = ""))

  # collect times, status, variables and reorder samples
  # to make the algorithm more clear to read and track
  cbind(event = unclass(Y)[,2], # 1 indicates event, 0 indicates cens
        times = unclass(Y)[,1],
        mf[, -1]) %>%
    arrange(times)
}
```

```
simulateCoxSGD(dCox, learningRates = function(x){1/(100*sqrt(x))},
               max.iter = 10, epsilon = 1e-5, beta_0 = c(2,2))
```

```
logitGD <- function(y, x, optim.method = "GDI", eps = 10e-4,
                    max.iter = 100, alpha = function(t){1/t}, beta_0 = c(0,0)){
  stopifnot(length(y) == length(x) & optim.method %in% c("GDI", "GDII", "SGDI")
            & is.numeric(c(max.iter, eps, x)) & all(c(eps, max.iter) > 0) &
              is.function(alpha))
  iter <- 0
  err <- list()
  err[[iter+1]] <- eps+1
  w_old <- beta_0

  res <-list()
  while(iter < max.iter && (abs(err[[ifelse(iter==0,1,iter)]]) > eps)){

    iter <- iter + 1
    if (optim.method == "GDI"){
      w_new <- w_old + alpha(iter)*updateWeightsGDI(y, x, w_old)
    }
    if (optim.method == "GDII"){
      w_new <- w_old + as.vector(inverseHessianGDII(x, w_old)%*%
                                   updateWeightsGDI(y, x, w_old))
    }
    if (optim.method == "SGDI"){
      w_new <- w_old + alpha(iter)*updateWeightsSGDI(y[iter], x[iter], w_old)
    }
    res[[iter]] <- w_new
```

```
    err[[iter]] <- sqrt(sum((w_new - w_old)^2))

    w_old <- w_new

  }
  return(list(steps = c(list(beta_0),res), errors = c(list(c(0,0)),err)))
}

updateWeightsGDI <- function(y, x, w_old){
  #(1/length(y))*c(sum(y-p(w_old, x)), sum(x*(y-p(w_old, x))))
  c(sum(y-p(w_old, x)), sum(x*(y-p(w_old, x))))
}

updateWeightsSGDI <- function(y_i, x_i, w_old){
  c(y_i-p(w_old, x_i), x_i*(y_i-p(w_old, x_i)))
}

p <- function(w_old, x_i){
  1/(1+exp(-w_old[1]-w_old[2]*x_i))
}

inverseHessianGDII <- function(x, w_old){
  solve(
    matrix(c(
      sum(p(w_old, x)*(1-p(w_old, x))),
      sum(x*p(w_old, x)*(1-p(w_old, x))),
      sum(x*p(w_old, x)*(1-p(w_old, x))),
      sum(x*x*p(w_old, x)*(1-p(w_old, x)))
    ),
    nrow =2 )
  )
}
set.seed(1283)
x <- runif(10000)
z <- 2 + 3*x
pr <- 1/(1+exp(-z))
y <- rbinom(10000,1,pr)


global_loglog <- function(beta1, beta2, xX, yY){
  sum(yY*(beta1+beta2*xX)-log(1+exp(beta1+beta2*xX)))
}



calculate_outer <- function(x, y){
  ## contours
  outer_res <- outer(seq(0,4, length = 100),
                     seq(0,5, length = 100),
                     Vectorize( function(beta1,beta2){
                       global_loglog(beta1, beta2, xX = x, yY = y)
                     } )
  )
```

```
  outer_res_melted <- melt(outer_res)


  outer_res_melted$Var1 <- as.factor(outer_res_melted$Var1)
  levels(outer_res_melted$Var1) <- as.character(seq(0,4, length = 100))
  outer_res_melted$Var2 <- as.factor(outer_res_melted$Var2)
  levels(outer_res_melted$Var2) <- as.character(seq(0,5, length = 100))
  outer_res_melted$Var1 <- as.numeric(as.character(outer_res_melted$Var1))
  outer_res_melted$Var2 <- as.numeric(as.character(outer_res_melted$Var2))
  return(outer_res_melted)
}



library(ggplot2); library(ggthemes); library(reshape2)
graphSGD <- function(beta, y, x, seed = 4561, outerBounds = calculate_outer(x,y)){
  set.seed(seed)

  beta <- rev(beta)

  logitGD(y, x, optim.method = "GDI",beta_0 = beta,
          eps = 10e-4, max.iter = 10000,
          alpha = function(t){1/(1000*sqrt(t))})$steps -> GDI.S

  logitGD(y, x, optim.method = "GDII", beta_0 = beta,
          eps = 10e-4, max.iter = 5000)$steps -> GDII

  ind2 <- sample(length(y))
  logitGD(y[ind2], x[ind2], optim.method = "SGDI", beta_0 = beta,
          max.iter = 10000, eps = 10e-4,
          alpha = function(t){1/sqrt(t)})$steps -> SGDI.1.S
  ind3 <- sample(length(y))
  logitGD(y[ind3], x[ind3], optim.method = "SGDI", beta_0 = beta,
          max.iter = 10000, eps = 10e-4,
          alpha = function(t){5/sqrt(t)})$steps -> SGDI.5.S
  ind4 <- sample(length(y))
  logitGD(y[ind4], x[ind4], optim.method = "SGDI", beta_0 = beta,
          max.iter = 10000, eps = 10e-4,
          alpha = function(t){6/sqrt(t)})$steps -> SGDI.6.S

  do.call(rbind, c(GDI.S, GDII, SGDI.1.S, SGDI.5.S, SGDI.6.S)) -> coeffs
  unlist(lapply(list(GDI.S, GDII, SGDI.1.S, SGDI.5.S, SGDI.6.S),
                length)) -> algorithm
  data2viz <- cbind(as.data.frame(coeffs),
  algorithm = unlist(mapply(rep,
                            c(paste("GDI", length(GDI.S), "steps"),
                            paste("GDII", length(GDII), "steps"),
                            paste("SGDI.1", length(SGDI.1.S), "steps"),
                            paste("SGDI.5", length(SGDI.5.S), "steps"),
                            paste("SGDI.6", length(SGDI.6.S), "steps")),
                           algorithm)))
  names(data2viz)[1:2] <- c("Intercept", "X")
  data2viz$algorithm <- factor(data2viz$algorithm, levels = rev(levels(data2viz$algorithm)))
```

```
  beta[2] -> XX
  beta[1] -> YY


  ggplot()+
    geom_path(aes(x = data2viz$X,
                  y = data2viz$Intercept,
                  col = data2viz$algorithm,
                  group = data2viz$algorithm), size = 1) +
    geom_point(aes(as.vector(round(coefficients(glm(y~x,
                           family = 'binomial')), 2)[2]),
                   as.vector(round(coefficients(glm(y~x,
                           family = 'binomial')), 2)[1])),
               col = "black", size = 4, shape = 15) +
    geom_point(aes(x=XX, y=YY),
               col = "black", size = 4, shape = 17) +
    theme_bw(base_size = 20) +
    theme(panel.border = element_blank(),
          legend.key = element_blank()) +
    scale_colour_brewer(palette="Set1", name = 'Algorithm') +
    xlab('X') +
    ylab('Intercept')  -> pl_g

  return(pl_g)
}
```

```
full_cox_loglik <- function(beta1, beta2, x1, x2, censored){
  sum(rev(censored)*(beta1*rev(x1) + beta2*rev(x2) -
                       log(cumsum(exp(beta1*rev(x1) + beta2*rev(x2))))))
}


calculate_outer_cox <- function(x1, x2, censored){
  ## contours
  outer_res <- outer(seq(-1,3, length = 100),
            seq(0,4, length = 100),
            Vectorize( function(beta1,beta2){
              full_cox_loglik(beta1, beta2, x1 = x1, x2 = x2, censored = censored)
            } )
  )
  outer_res_melted <- melt(outer_res)
  outer_res_melted$Var1 <- as.factor(outer_res_melted$Var1)
  levels(outer_res_melted$Var1) <- as.character(seq(-1,3, length = 100))
  outer_res_melted$Var2 <- as.factor(outer_res_melted$Var2)
  levels(outer_res_melted$Var2) <- as.character(seq(0,4, length = 100))
  outer_res_melted$Var1 <- as.numeric(as.character(outer_res_melted$Var1))
  outer_res_melted$Var2 <- as.numeric(as.character(outer_res_melted$Var2))
  return(outer_res_melted)
}
simulateCoxSGD <- function(dCox = dCox, learningRates = function(x){1/x},
                       epsilon = 1e-03, beta_0 = c(0,0), max.iter = 100){

  sample(1:90, size = 10^4, replace = TRUE) -> group
```

```
split(dCox, group) -> dCox_splitted
coxphSGD(Surv(time, status)~x.1+x.2, data = dCox_splitted,
         epsilon = epsilon, learningRates = learningRates,
         beta_0 = beta_0, max.iter = max.iter*90) -> estimates

sample(1:60, size = 10^4, replace = TRUE) -> group
split(dCox, group) -> dCox_splitted
coxphSGD(Surv(time, status)~x.1+x.2, data = dCox_splitted,
         epsilon = epsilon, learningRates = learningRates,
         beta_0 = beta_0, max.iter = max.iter*60) -> estimates2

sample(1:120, size = 10^4, replace = TRUE) -> group
split(dCox, group) -> dCox_splitted
coxphSGD(Surv(time, status)~x.1+x.2, data = dCox_splitted,
         epsilon = epsilon, learningRates = learningRates,
         beta_0 = beta_0, max.iter = max.iter*120) -> estimates3

sample(1:200, size = 10^4, replace = TRUE) -> group
split(dCox, group) -> dCox_splitted
coxphSGD(Surv(time, status)~x.1+x.2, data = dCox_splitted,
         epsilon = epsilon, learningRates = learningRates,
         beta_0 = beta_0, max.iter = max.iter*200) -> estimates4


sample(1:30, size = 10^4, replace = TRUE) -> group
split(dCox, group) -> dCox_splitted
coxphSGD(Surv(time, status)~x.1+x.2, data = dCox_splitted,
         epsilon = epsilon, learningRates = learningRates,
         beta_0 = beta_0, max.iter = max.iter*30) -> estimates5


sample(1:10, size = 10^4, replace = TRUE) -> group
split(dCox, group) -> dCox_splitted
coxphSGD(Surv(time, status)~x.1+x.2, data = dCox_splitted,
         epsilon = epsilon, learningRates = learningRates,
         beta_0 = beta_0, max.iter = max.iter*10) -> estimates6

t(simplify2array(estimates$coefficients)) %>%
  as.data.frame() -> df1
t(simplify2array(estimates2$coefficients)) %>%
  as.data.frame() -> df2
t(simplify2array(estimates3$coefficients)) %>%
  as.data.frame() -> df3
t(simplify2array(estimates4$coefficients)) %>%
  as.data.frame() -> df4
t(simplify2array(estimates5$coefficients)) %>%
  as.data.frame() -> df5
t(simplify2array(estimates6$coefficients)) %>%
  as.data.frame() -> df6

df1 %>%
  mutate(version = paste("90 batches,", nrow(df1), " steps")) %>%
  bind_rows(df2 %>%
```

```
                mutate(version = paste("60 batches,", nrow(df2), " steps"))) %>%
    bind_rows(df3 %>%
                mutate(version = paste("120 batches,", nrow(df3), " steps"))) %>%
    bind_rows(df4 %>%
                mutate(version = paste("200 batches,", nrow(df4), " steps"))) %>%
    bind_rows(df5 %>%
                mutate(version = paste("30 batches,", nrow(df5), " steps"))) %>%
    bind_rows(df6 %>%
                mutate(version = paste("10 batches,", nrow(df6), " steps"))) -> d2ggplot

  return(list(d2ggplot = d2ggplot, est1 = estimates, est2 = estimates2,
              est3 = estimates3, est4 = estimates4, est5 = estimates5))

}
simulateCoxSGD(dCox, learningRates = function(x){1/(100*sqrt(x))},
               max.iter = 10, epsilon = 1e-5) -> d2ggplot
d2ggplot -> backpack
d2ggplot <- d2ggplot$d2ggplot
beta_0 = c(0,0)
solution = c(1,3)

pdf(file = "b_0_0_iter_10_e-5_100sqrt_878.pdf", width = 10, height = 10)
ggplot() +
  stat_contour(aes(x=outerCox$Var1,
                   y=outerCox$Var2,
                   z=outerCox$value),
               bins = 40, alpha = 0.25) +
  geom_path(aes(d2ggplot$V1, d2ggplot$V2, group = d2ggplot$version,
                colour = d2ggplot$version), size = 1) +
  theme_bw(base_size = 20) +
  theme(panel.border = element_blank(),
        legend.key = element_blank(), legend.position = "top") +
  scale_colour_brewer(palette="Dark2", name = 'Algorithm \n & Steps') +
  geom_point(aes(x = beta_0[1], y = beta_0[2]), col = "black", size = 4, shape = 17) +
  geom_point(aes(x = solution[1], y = solution[2]), col = "black", size = 4, shape = 15) +
  xlab("X1") + ylab("X2") +
  guides(col = guide_legend(ncol = 3))
dev.off()
```

```
extractSurvival <- function(cohorts){

  survivalData <- list()
  for(i in cohorts){
    get(paste0(i, ".clinical"), envir = .GlobalEnv) %>%
                select(patient.bcr_patient_barcode,
                             patient.vital_status,
                             patient.days_to_last_followup,
                             patient.days_to_death ) %>%
                mutate(bcr_patient_barcode = toupper(patient.bcr_patient_barcode),
                       patient.vital_status = ifelse(patient.vital_status %>%
                                                       as.character() =="dead",1,0),
                   barcode = patient.bcr_patient_barcode %>%
                                    as.character(),
```

```
                  times = ifelse( !is.na(patient.days_to_last_followup),
                       patient.days_to_last_followup %>%
                         as.character() %>%
                         as.numeric(),
                 patient.days_to_death %>%
                         as.character() %>%
                         as.numeric() )
                      ) %>%
   filter(!is.na(times)) -> survivalData[[i]]


  }
  do.call(rbind,survivalData) %>%
    select(bcr_patient_barcode, patient.vital_status, times) %>%
    unique

}


extractMutations <- function(cohorts, prc){
  mutationsData <- list()
  for(i in cohorts){
    get(paste0(i, ".mutations"), envir = .GlobalEnv) %>%
      select(Hugo_Symbol, bcr_patient_barcode) %>%
      filter(nchar(bcr_patient_barcode)==15) %>%
      filter(substr(bcr_patient_barcode, 14, 15)=="01") %>%
      unique -> mutationsData[[i]]
  }
  do.call(rbind,mutationsData) %>% unique -> mutationsData

  mutationsData %>%
    group_by(Hugo_Symbol) %>%
    summarise(count = n()) %>%
    arrange(desc(count)) %>%
    mutate(count_prc = count/length(unique(mutationsData$bcr_patient_barcode))) %>%
    filter_(paste0("count_prc > ",prc)) %>%
    select(Hugo_Symbol) %>%
    unlist -> topGenes

  mutationsData %>%
    filter(Hugo_Symbol %in% topGenes) -> mutationsData_top


  mutationsData_top %>%
    dplyr::group_by(bcr_patient_barcode) %>%
    dplyr::summarise(count = n()) %>%
    group_by(count) %>%
    summarise(total = n()) %>%
    arrange(desc(count))
#
#   mutationsData_top %>%
#     spread(Hugo_Symbol, bcr_patient_barcode) -> mutationsData_top_sp
```

```
  as.data.table(mutationsData_top) -> mutationsData_top_DT
  dcast.data.table(mutationsData_top_DT, bcr_patient_barcode ~ Hugo_Symbol , fill = 0) %>%
    as.data.frame -> mutationsData_top_dcasted

  mutationsData_top_dcasted[,-1][mutationsData_top_dcasted[,-1] != "0"] <- 1

  mutationsData_top_dcasted -> result
  names(result) <- gsub(names(result),pattern = "-",  replacement = "")
  result
}


extractCohortIntersection <- function(){

  data(package = "RTCGA.mutations")$results[,3] %>%
    gsub(".mutations", "", x = .) -> mutations_data
  data(package = "RTCGA.clinical")$results[,3] %>%
    gsub(".clinical", "", x = .) -> clinical_data

  intersect(mutations_data, clinical_data)
}

prepareCoxDataSplit <- function(mutationsData, survivalData, groups, seed = 4561){
  mutationsData %>%
  mutate(bcr_patient_barcode = substr(bcr_patient_barcode,1,12)) %>%
  left_join(survivalData,
            by = "bcr_patient_barcode") -> coxData

  coxData <- coxData[, -c(1,2)]

  coxData %>%
    filter(times > 0) %>%
    filter(!is.na(times)) -> coxData

  apply(coxData[,-c(1092, 1093)], MARGIN = 2,function(x){
    as.numeric(as.character(x))
  }) -> coxData[,-c(1092, 1093)]

  set.seed(seed)
  sample(groups, replace = TRUE, size = 6085) -> groups
  split(coxData, groups) #coxData_split
}

prepareForumlaSGD <- function(coxData){
  as.formula(paste("Surv(times, patient.vital_status) ~ ",
                   paste(names(coxData[[1]])[-c(1092, 1093)],
                         collapse="+"), collapse = ""))
}


full_cox_loglik_matrix <- function(beta, x, censored){
  order(x$times) -> order2
  x[order2, ] -> xORD
```

```
  censored[order2] -> censORD
  sum(censORD*(beta%*%x[, -which(names(x)=='times')] -
                       log(cumsum(exp(beta1*rev(x1) + beta2*rev(x2))))))
}



library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(RTCGA.clinical)
```

```
Loading required package: RTCGA
Loading required package: knitr
Welcome to the RTCGA (version: 1.1.10).
```

```
library(RTCGA.mutations)
library(data.table)
```

```
Attaching package: 'data.table'

The following objects are masked from 'package:dplyr':

    between, last
```

```
library(coxphSGD)
```

```
Loading required package: survival

Attaching package: 'coxphSGD'

The following object is masked _by_ '.GlobalEnv':

    dataCox
```

Do analizy badającej wpływ występowania mutacji genów na czas przeżycia wykorzystano dane kliniczne i dane o występujących u pacjentów mutacjach genetycznych. Starano się wykorzystać dane ze wszystkich 38 dostępnych kohort nowotworowych z badania *The Cancer Genome Atlas* (TCGA), jednak nie dla wszystkich kohort umieszczono w badaniu dane o mutacjach. Częśc wspólną nazw dla kohort zawierających zarówno dane kliniczne oraz dane o mutacjach wygenerowaną dzięki wywołaniu

```
(extractCohortIntersection() -> cohorts)
```

```
 [1] "ACC"      "BLCA"     "BRCA"     "CESC"     "CHOL"     "COAD"
 [7] "COADREAD" "DLBC"     "ESCA"     "GBM"      "GBMLGG"   "HNSC"
[13] "KICH"     "KIPAN"    "KIRC"     "KIRP"     "LAML"     "LGG"
[19] "LIHC"     "LUAD"     "LUSC"     "OV"       "PAAD"     "PCPG"
[25] "PRAD"     "READ"     "SARC"     "SKCM"     "STAD"     "STES"
[31] "TGCT"     "THCA"     "UCEC"     "UCS"      "UVM"
```

Następnie dla tak otrzymanych 35 kohort nowotworowych uzyskano dane o statusie pacjenta (śmierć bądź cenzurowanie) oraz jego czasie spędzonym pod obseracją dzięki funkcji

```
head(extractSurvival(cohorts) -> survivalData)
```

```
      bcr_patient_barcode patient.vital_status times
ACC.1        TCGA-OR-A5J1                   1  1355
ACC.2        TCGA-OR-A5J2                   1  1677
ACC.3        TCGA-OR-A5J3                   0  1942
ACC.4        TCGA-OR-A5J4                   1   423
ACC.5        TCGA-OR-A5J5                   1   365
ACC.6        TCGA-OR-A5J6                   0  2428
```

Dane o mutacjach występujących wśród tkanek nowotworowych kolejnych pacjentów uzyskano za pomocą

```
extractMutations(cohorts, 0.02) -> mutationsData
```

```
Using 'bcr_patient_barcode' as value column. Use 'value.var' to override
```

```
mutationsData[1:6, c(1,4,56,100,207,801)]
```

```
  bcr_patient_barcode A2ML1 ALMS1 ATP2B2 CNTNAP4 PLEC
1     TCGA-02-0003-01     0     1      0       0    0
2     TCGA-02-0033-01     0     0      0       0    0
3     TCGA-02-0047-01     0     0      0       0    0
4     TCGA-02-0055-01     0     0      0       0    0
5     TCGA-02-2470-01     0     0      0       0    0
6     TCGA-02-2483-01     0     0      1       0    0
```

gdzie wybrano jedynie te geny, których mutacja dotyczyła co najmniej 2 % pacjentów mających zarówno dane kliniczne jak i dane o występujących mutacjach w genach.

Dla tak otrzymanych dwóch zbiorów danych połączono dla pacjentów informacje kliniczne z informacjami o mutacjach dzięki przypisanym do pacjentów i ich próbek kodów `bcr_patient_barcode`, by ostatecznie podzielić zbiór pacjentów na 100 losowo utworzonych grup.

```
set.seed(4561)
prepareCoxDataSplit(mutationsData,survivalData, groups = 100) -> coxData_split
head(coxData_split[[1]][c(1,10), c(210,302,356,898,911,1092:1093)])
```

```
     COL14A1 DOCK9 FASN SEMA5A SHPRH patient.vital_status times
81         0     0    0      0     0                   1     7
1068       1     0    0      1     0                   1  1171
```

Niezbędną formułę modelu potrzebną do sprezycowania, które geny (a pozostało ich 1091) należy uwzględnić w modelu uzyskano dzięki pomocniczej funkcji

```
prepareForumlaSGD(coxData_split) -> formulaSGD
```

Ostatecznie dla 6085 pacjentów, którzy posiadali informacje o występujących mutacjach, oraz dla których odnotowano komplet i poprawność danych klinicznych dotyczących statusu i obserwowanego czasu przeżycia wyliczono współczynniki modelu proporcjonalnych hazardów Coxa z wykorzystaniem stochastycznego spadku gradientu do estymacji. Model dopasowano wielokrotnie z różnymi ciągami odpowiadającymi za długość kroku algorytmu, dodatkowo badano różną ilość epok w algorytmie. Dla tak powstałych kilku modeli wybrano ten, który dla swoich współczynników dawał największą wartość funkcji częściowej log-wiarogodności dla niewykorzystanej do uczenia próbki, zawierającej 2 ostatnie zaobserwowane podzbiory obserwacji. Dla każdego z ciągów $1/t, 1/50 * \sqrt(t), 100/5 * \sqrt(100)$ odpowiadających długościom kroków w algorytmie wyznaczono współczynniki modelu dla 5 epok, dzięki czemu możliwe było rozważanie postępu danego wariantu algorytmu również po 1, 2, 3 czy 4 epokach.

```
coxData_split[99:100] -> testCox
coxData_split[1:98] -> trainCox
coxphSGD(formulaSGD, data = trainCox, max.iter = 490) -> model_1_over_t
coxphSGD(formulaSGD, data = trainCox, max.iter = 490,
         learningRates = function(t){1/(50*sqrt(t))}) -> model_1_over_50sqrt_t
coxphSGD(formulaSGD, data = trainCox, max.iter = 490,
         learningRates = function(t){1/(100*sqrt(t))}) -> model_1_over_100sqrt_t
```

Niemożliwe było sprawdzenie założeń modelu dotyczących proporcjonalności hazardu, gdyż zakładano napływającą postać danych (stąd podział danych na 100 grup). Dla takiej postaci pojawiania się danych ciężko także mówić o jakiejkolwiek diagnostyce poprawności dopasowania modelu i dokładności otrzymanych wpsółczynników. Nie stworzono teorii pozwalającej badać istotność statystyczną otrzymanych współczynników w modelu, jednak założono, że współczynniki dostatecznie odległe od 0 można uznać za istotnie wpływające na czas życia pacjenta. Współczynniki dodatnie oznaczają zwiększenie hazardu pacjenta posiadającego mutację w danym genie w stosunku do pacjentów nie posiadających mutacji w danym genie. Współczynniki ujemne oznaczają zmniejszenie hazardu pacjenta posiadającego mutację w danym genie w stosunku do pacjentów nie posiadających mutacji w danym genie. Wzrost proporcji hazardu można otrzymać dla danego genu poprzez obłożenie współczynnika funkcją wykładniczą o wykładniku e.

Wyniki estymacji dla genów zawierających największe co do modułu współczynniki można znaleźć w Tabeli 1.

```r
coxphSGD <- function(formula, data, learningRates = function(x){1/x},
                     beta_0 = 0, epsilon = 1e-5, max.iter = 500 ) {
  checkArguments(formula, data, learningRates,
                 beta_0, epsilon) -> beta_start # check arguments
  n <- length(data)
  diff <- epsilon + 1
  i <- 1
  beta_new <- list()     # steps are saved in a list so that they can
  beta_old <- beta_start # be traced in the future
  # estimate
  while(i <= max.iter & diff > epsilon) {
    beta_new[[i]] <- coxphSGD_batch(formula = formula, beta = beta_old,
        learningRate = learningRates(i), data = data[[ifelse(i%%n==0,n,i%%n)]]) %>%
      unlist
    diff <- sqrt(sum((beta_new[[i]] - beta_old)^2))
    beta_old <- beta_new[[i]]
    i <- i + 1  ; cat("\r iteration: ", i, "\r")
  }  # return results
  list(Call = match.call(), epsilon = epsilon, learningRates = learningRates,
       steps = i, coefficients = c(list(beta_start), beta_new))
}

coxphSGD_batch <- function(formula, data, learningRate, beta){
  # collect times, status, variables and reorder samples
  # to make the algorithm more clear to read and track
  batchData <- prepareBatch(formula = formula, data = data)
  # calculate the log-likelihood for this batch sample
  partial_sum <- list()
  foreach(k = 1:nrow(batchData)) %do% {
    # risk set for current time/observation
    risk_set <- batchData %>% filter(times >= batchData$times[k])

    nominator <- apply(risk_set[, -c(1,2)], MARGIN = 1, function(element){
      element * exp(element * beta)
    }) %>% rowSums()

    denominator <- apply(risk_set[, -c(1,2)], MARGIN = 1, function(element){
      exp(element * beta)
    }) %>% rowSums()

    partial_sum[[k]] <-
      batchData[k, "event"] * (batchData[k, -c(1,2)] - nominator/denominator)
  }
  do.call(rbind, partial_sum) %>%
    colSums() -> U_batch

  return(beta + learningRate * U_batch)
}
```