Untitled

Marcin Kosinski

25.10.2015

dupa sraka `logitGD()` jasd

```
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))))
}

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.

                                # estymacja
while(i < n | diff < eps) do            # do zbieżności lub wyczerpania zbiorów

  batch = data[[i]]

  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
```