527 Perceptron algorithm

Xinying Fang

11/3/2021

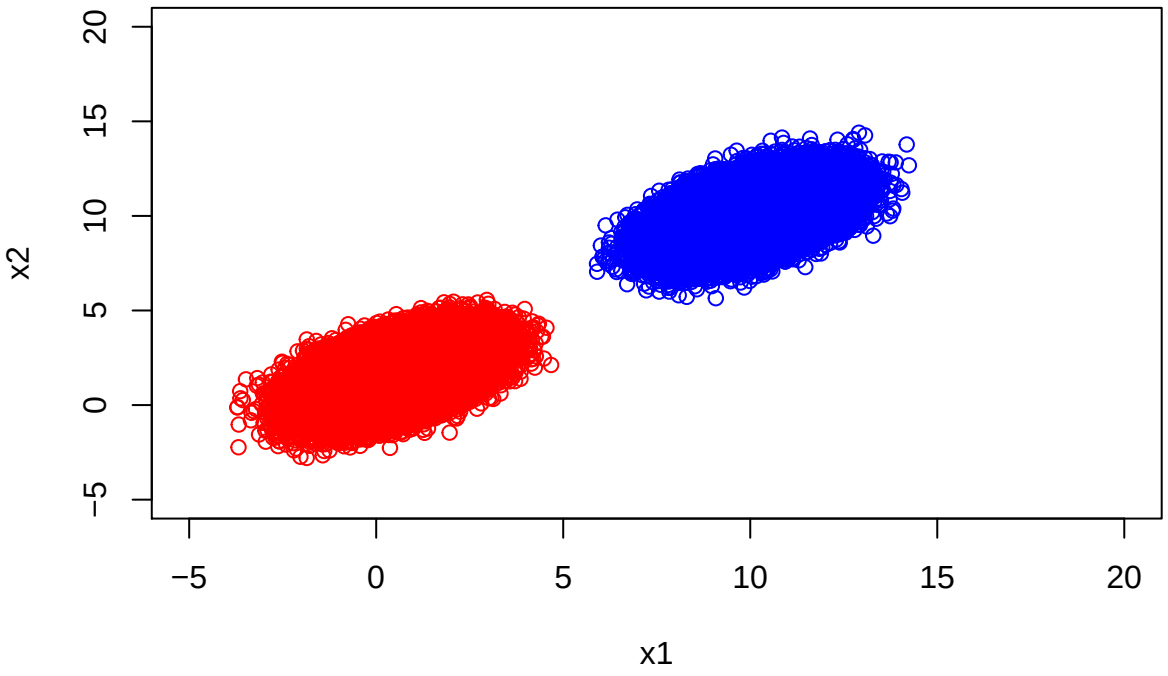
```
library(mvtnorm)
```

Simulate data

To apply the perceptron algorithm, we simulate two groups of data where the separating hyper-plane can perfectly separate them:

```
n=100000
X = rmvnorm(n, mean=c(0.5,1.5), sigma=matrix(c(1,0.5,0.5,1), ncol = 2))
x1 = X[,1]
x2 = X[,2]
X = rmvnorm(n, mean=c(10,10), sigma=matrix(c(1,0.5,0.5,1), ncol = 2))
x11 = X[,1]
x22 = X[,2]

p = plot(x1,x2,col="red",xlim=c(-5,20),ylim=c(-5,20))+ points(x11,x22,col="blue")
```



Gradient descent

For both GD and SGD, we run the algorithm for k times and in each run, the simulated data contains n data points.

```r
k = 100
n = 100000

rho = 0.5 # step size is set to be 0.5

converge <- c()
start_time <- Sys.time()
for (i in 1:k){
  X = rmvnorm(n, mean=c(0.5,1.5), sigma=matrix(c(1,0.5,0.5,1), ncol = 2))
  x1 = X[,1]
  x2 = X[,2]
  X = rmvnorm(n, mean=c(10,10), sigma=matrix(c(1,0.5,0.5,1), ncol = 2))
  x11 = X[,1]
  x22 = X[,2]

  X <- matrix(c(x1,x11,x2,x22),ncol=2)
  X <- cbind(rep(1,nrow(X)), X)

  Y <- matrix(rep(c(-1,1),each=n), ncol=1)

  j = 1
  bmat <- matrix(c(1,1,1),ncol=1) # initial bmat = matrix(b0,b1,b2,ncol=1)
  m = 10000
  while (j <= m){

    xb <- X%*%bmat
    # ypred <- ifelse(xb <=0, -1, 1)
    # Y * xb
    if (all((Y * xb) > 0)==FALSE){
      misind <- which(Y * xb < 0)
      ### update
      Ymis <- matrix(Y[misind,], ncol=1)
      Xmis <- matrix(X[misind,], ncol=3)

      bmat.new <- bmat + rho * (t(Xmis) %*% Ymis)
      bmat <- bmat.new
      j = j+1
      if (j == m+1){
        converge <- c(converge, 0)
      }
    } else {
      # print(bmat)
      print(j)
      converge <- c(converge, 1)
      break
    }
  }
}
end_time <- Sys.time()
end_time - start_time
mean(converge)
```

Stochastic gradient descent

```
# judge = FALSE
converge_sgd <- c()
start_time <- Sys.time()
for (i in 1:k){

  X = rmvnorm(n, mean=c(0.5,1.5), sigma=matrix(c(1,0.5,0.5,1), ncol = 2))
  x1 = X[,1]
  x2 = X[,2]
  X = rmvnorm(n, mean=c(10,10), sigma=matrix(c(1,0.5,0.5,1), ncol = 2))
  x11 = X[,1]
  x22 = X[,2]

  X <- matrix(c(x1,x11,x2,x22),ncol=2)
  X <- cbind(rep(1,nrow(X)), X)

  Y <- matrix(rep(c(-1,1),each=n), ncol=1)

  rho = 0.5

  j = 1
  bmat <- matrix(c(1,1,1),ncol=1) # initial bmat = matrix(b0,b1,b2,ncol=1)
  m = 10000
  while (j <= m){

    xb <- X%*%bmat
    # ypred <- ifelse(xb <=0, -1, 1)
    # Y * xb
    if (all((Y * xb) > 0)==FALSE){
      misind <- which(Y * xb < 0)
      misind <- sample(misind,1)
      ### update
      Ymis <- matrix(Y[misind,], ncol=1)
      Xmis <- matrix(X[misind,], ncol=3)

      bmat.new <- bmat + rho * (t(Xmis) %*% Ymis)
      bmat <- bmat.new
      j = j+1
      if (j == m+1){
        converge_sgd <- c(converge_sgd, 0)
      }
    } else {
      # print(bmat)
      # print(j)
      converge_sgd <- c(converge_sgd, 1)
      break
    }
  }
}
end_time <- Sys.time()
end_time - start_time
mean(converge_sgd)
```

When sample size is small (e.g. n=100, 1000), both GD and SGD are easy to converge. The convergence rates for them are 100% among k = 10,000 times of running.

When sample size is large (e.g. n=100,000), GD fails to give good convergence. Among 100 times of running, the convergence rate for GD is about 30%, while the convergence rate for SGD is around 100%. The time for convergence is also fast for SGD, taking less than 1 sec to converge for each running.

Possible reasons for the large discrepancy between GD and SGD when sample size is large:

- Large data set takes GD more time to consider all mis-classified data points at one iteration, especially when my starting hyper-plane is far from the "correct" hyper-plane.
- GD can converge for 100%, but takes more iterations.
- Update in each iteration for GD is slow. If we want faster convergence, increasing both maximum iteration and step size may be one solution, but not a decisive one for GD to catch up with SGD.