PHS 597 – Homework 3 – Perceptron Algorithm – Fall 2021

Havell Markus, hmm5304@psu.edu

September 6, 2021

Assginment description

- The goal of this assignment is to implement the **Perceptron Learning Algorithm** using normal gradient descent (NGD) and stochastic gradient descent (SGD).

Perceptron Learning Algorithm

- The perceptron learning algorithm tries to find a hyperplane that separates points in different classes by minimizing the distance of misclassified points to the decision boundary.
- The goal is to minimize Equation 1, M are the indexes for the misclassified points.
- The update rule is based on partial derivatives of β and β_0.
- For NGD, I will update β and β_0 by using all the misclassified points from M, as shown in Equation 2 and 3.
- For SGD, I will update β and β_0 by randomly selecting one of the misclassified points from M, as shown in Equation 4 and 5.

$$D(\beta, \beta_0) = -\sum_{i \in M} y_i(x_i^T \beta + \beta_0) \tag{1}$$

$$\beta_{new} \leftarrow \beta_{old} + \rho * \sum_{i \in M} (y_i * x_i) \tag{2}$$

$$\beta_{0,new} \leftarrow \beta_{0,old} + \rho * \sum_{i \in M} (y_i) \tag{3}$$

$$\beta_{new} \leftarrow \beta_{old} + \rho * (y_i * x_i) \tag{4}$$

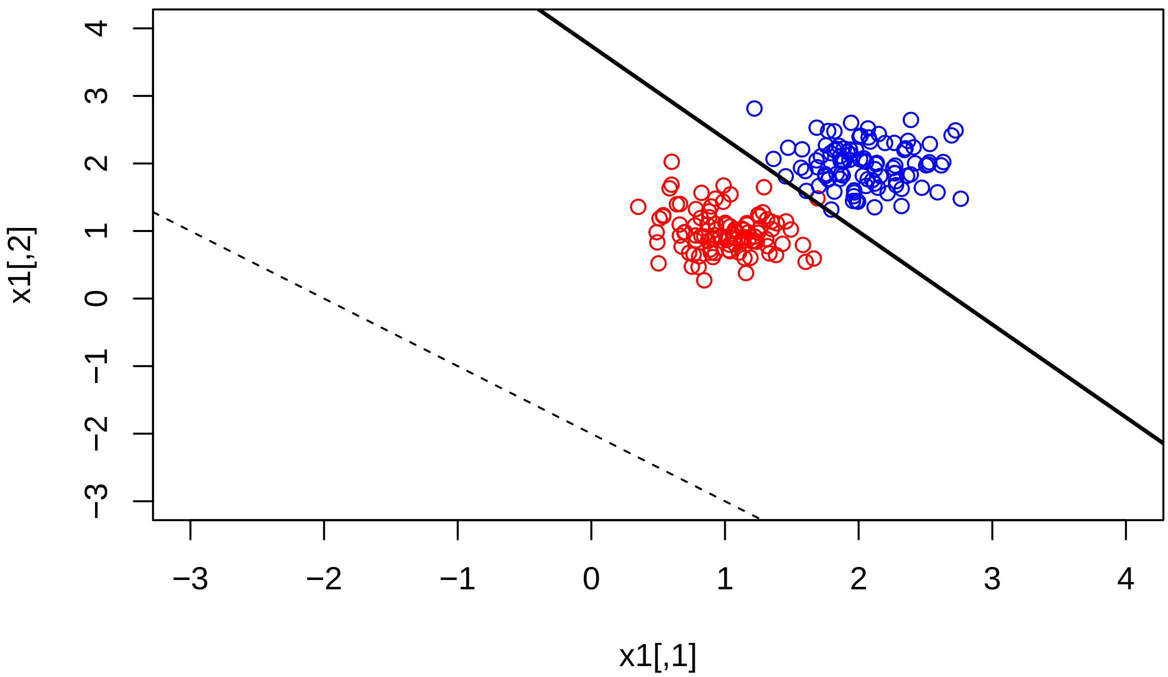
$$\beta_{0,new} \leftarrow \beta_{0,old} + \rho * (y_i) \tag{5}$$

Implementing Normal Gradient Descent

```
library(MASS)
library(mvtnorm)
```

- The hyperplane misses one point

```r
set.seed(123)
x1 <- rmvnorm(100,mean=c(1,1),sigma=0.1*diag(2))
x2 <- rmvnorm(100,mean=c(2,2),sigma=0.1*diag(2))
plot(x1,xlim=c(-3,4),ylim=c(-3,4),col='red')
points(x2,col='blue')
x <- cbind(rep(1,200), rbind(x1,x2))

y <- matrix(0, nrow=200, ncol=1);
y[1:100,1] <- 1
y[101:200,1] <- -1

beta <- matrix(c(1,0.5,0.5))

abline(-beta[1]/beta[3],-beta[2]/beta[3], lty = 2)

y.pred <- y*(x %*% beta)
y.miss <- which(y.pred < 0)

threshold <- 0.0001
lambda <- 0.001
t <- 0

while(length(y.miss) > 0 & t < 1000){

  beta_new <- beta + t(lambda*(t(y[y.miss,]) %*% x[y.miss, ]))
  beta <- beta_new
  t <- t + 1

  y.pred <- y*(x %*% beta)
  y.miss <- which(y.pred < 0)

}

abline(-beta[1]/beta[3],-beta[2]/beta[3], lwd = 2)
```




```
length(y.miss)
```

```
## [1] 1
```

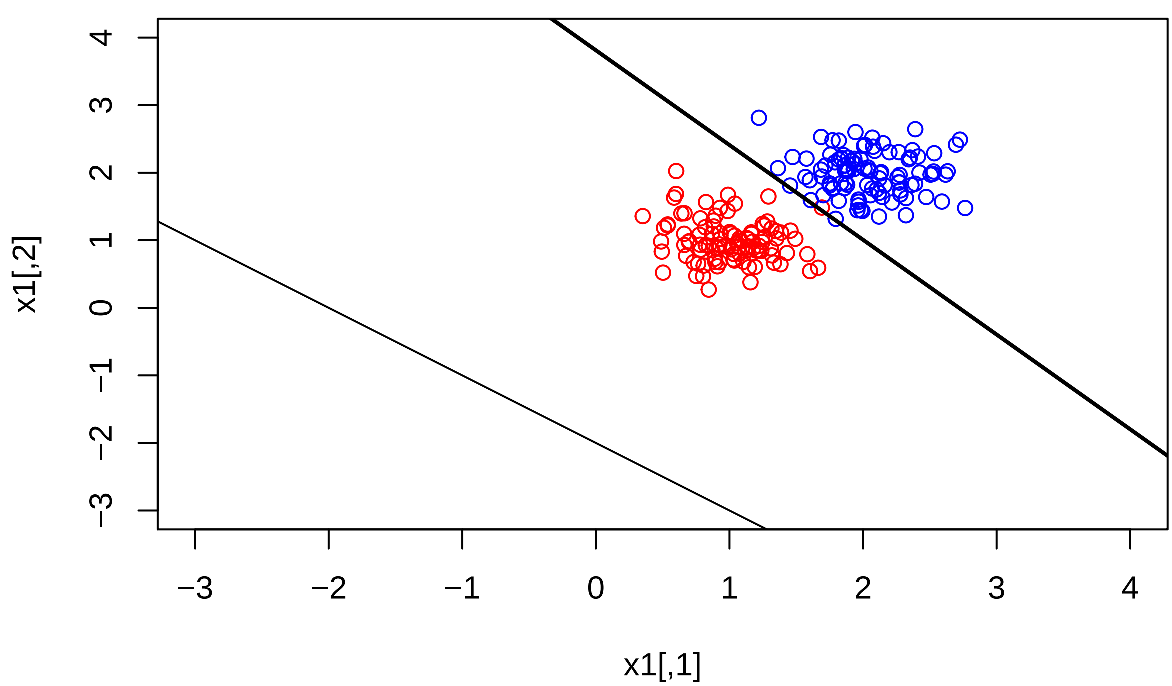
```
beta_new
```

```
##            [,1]
## [1,]  0.6690000
## [2,] -0.2459040
## [3,] -0.1789714
```

Implementing Stochastic Gradient Descent

```
plot(x1,xlim=c(-3,4),ylim=c(-3,4),col='red')
points(x2,col='blue')

beta <- matrix(c(1,0.5,0.5))

abline(-beta[1]/beta[3],-beta[2]/beta[3]);

y.pred <- y*(x %*% beta)
y.miss <- which(y.pred < 0)
```

```r
threshold <- 0.0001
lambda <- 0.001
t <- 0

while(length(y.miss) > 0 & t < 1000){

  ith_miss <- sample(y.miss, size = 1)

  beta_new <- beta + t(lambda*(t(y[ith_miss,]) %*% x[ith_miss, ]))
  beta <- beta_new
  t <- t + 1

  y.pred <- y*(x %*% beta)
  y.miss <- which(y.pred < 0)

}

abline(-beta[1]/beta[3],-beta[2]/beta[3], lwd = 2)
```




```r
length(y.miss)
```

```
## [1] 1
```

```r
beta_new
```

```
##            [,1]
## [1,]  0.7020000
## [2,] -0.2582777
## [3,] -0.1841264
```