Perceptron Method in Classification

Perceptron method is the foundation of SVM. It is easier to code becasue it applies only to linearly separable two classes.But the idea of hyperplane is more clear in this implementation

To implement perceptron, first we need to generate two linearly separable classes.This is done by a function called fakedata

```
library("MASS")
fakedata <- function(w, n) {
    d <- length(w)
    class1 <- matrix(ncol = d, nrow = n)
    class2 <- matrix(ncol = d, nrow = n)
    a <- 1
    b <- 1
    for (i in 1:1e+05) {
        x <- c(rnorm(d - 1), 1)
        if (a <= n && t(w) %*% x < 0) {
            class1[a, ] <- x
            a <- a + 1
        }
        if (b <= n && t(w) %*% x > 0) {
            class2[b, ] <- x
            b = b + 1
        }
    }
    Result <- list(class1, class2)
    names(Result) <- c("-1", "1")
    Result
}
```

Then I need a evaluation function to evaluate the classification hyperplane.

```
Evaluation <- function(S, z) {
    class1 <- S[[1]]
    class2 <- S[[2]]
    n <- nrow(class1)
    y1 <- matrix(ncol = n, nrow = 1)
    y2 <- matrix(ncol = n, nrow = 1)
    for (i in 1:n) {
        x1 <- class1[i, ]
        x2 <- class2[i, ]
        if (t(z) %*% x1 < 0) {
            y1[i] <- -1
        } else {
            y1[i] <- 1
        }
        if (t(z) %*% x2 > 0) {
            y2[i] <- 1
        } else {
            y2[i] <- -1
        }
    }
    Result <- list(y1, y2)
    names(Result) <- c("-1", " 1")
    return(Result)
}
```

Implement two versions of the Perceptron training algorithm.

1. The batch version with variable step size alpha(k) = 1/k ,where k is the number of the current iteration.

2. The fixed-increment single sample version

```
p1 <- function(S, y) {
    n <- nrow(S[[1]])
    m <- ncol(S[[1]])
    R <- matrix(ncol = m, nrow = 1)
    z <- rnorm(m)
    R[1, ] <- z
    for (k in 1:10000) {
        R <- rbind(R, matrix(ncol = m, nrow = 1))
        Cp <- 0
        dCp <- 0
        for (i in 1:n) {
            x1 <- S[[1]][i, ]
            x2 <- S[[2]][i, ]
            if (z %*% x1 > 0) {
                Cp <- Cp + abs(z %*% x1)
                dCp <- dCp + (-y[[1]][i]) * x1
            }
            if (z %*% x2 < 0) {
                Cp <- Cp + abs(z %*% x2)
                dCp <- dCp + (-y[[2]][i]) * x2
            }
        }
        if (Cp == 0) {
            return(R[-(k + 1), ])
        }
        z <- z - (1/k) * dCp
        R[k + 1, ] <- z
    }
    return(k)
}
```

```
p2 <- function(S, y) {
    n <- nrow(S[[1]])
    m <- ncol(S[[1]])
    R <- matrix(ncol = m, nrow = 1)
    z <- rnorm(m)
    R[1, ] <- z
    for (k in 1:10000) {
        Cp <- 0
        R <- rbind(R, matrix(ncol = m, nrow = 1))
        for (i in 1:n) {
            x1 <- S[[1]][i, ]
            x2 <- S[[2]][i, ]
            if (t(z) %*% x1 > 0) {
                z = z - x1
                Cp <- Cp + abs(z %*% x1)
            }
            if (t(z) %*% x2 < 0) {
                z = z + x2
                Cp <- Cp + abs(z %*% x2)
            }
            R[k + 1, ] <- z
        }
        if (Cp == 0) {
            return(R[-(k + 1), ])
        }
    }
    return(k)
}
```

use these two methods to train hyperplane

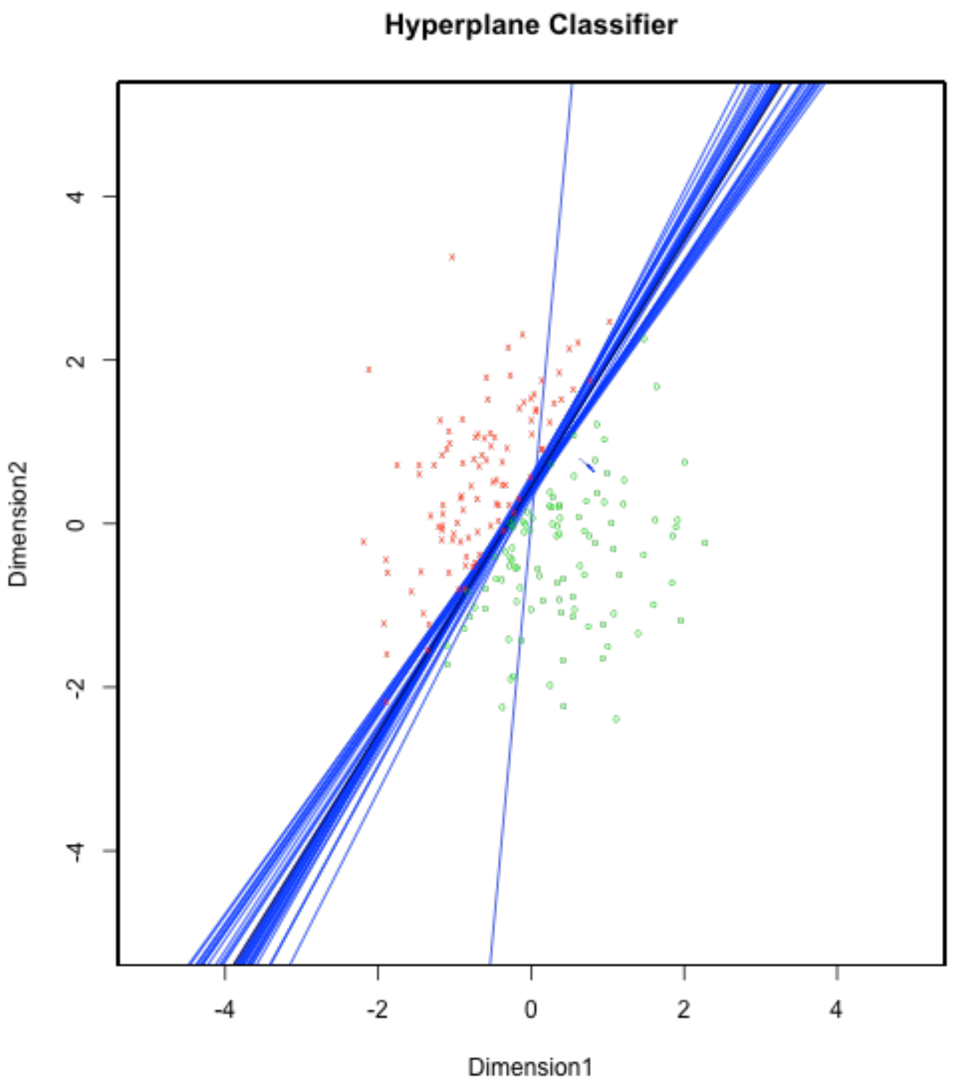
```
w <- rnorm(3)
train <- fakedata(w, 100)
test <- fakedata(w, 100)
y <- Evaluation(train, w)

r1 <- p1(train, y)
r2 <- p2(train, y)
```

Finally, visulization of hyperplane and affine hyperplane

I write a plot function

```
plot.hyperplane <- function(rx) {
    n <- nrow(rx)
    for (i in 1:n) {
        m <- (1/(sqrt(rx[i, ][1]^2 + rx[i, ][2]^2))) * rx[i, ]
        r <- Null(m[1:2])
        z <- r + m[3] * c(m[1], m[2])
        if (i < n) {
            b = r[2]/r[1]
            plot(z[2] ~ z[1], xlim = c(-5, 5), ylim = c(-5,
5), pch = 19, cex = 0.001,
                xaxt = "n", yaxt = "n", ann = F, col = 4)
            abline(a = -sign(m[2]) * m[3] * sqrt(b^2 + 1), b =
r[2]/r[1], col = 4)
            par(new = T)
        } else {
            plot(z[2] ~ z[1], xlim = c(-5, 5), ylim = c(-5,
5), pch = 19, cex = 0.001,
                xlab = "Dimension1", ylab = "Dimension2", main
= "Hyperplane Classifier",
                col = 4)
            b = r[2]/r[1]
            abline(a = -sign(m[2]) * m[3] * sqrt(b^2 + 1), b =
r[2]/r[1], col = 1)
        }
    }
    points(test[[1]][, -3], pch = c("x"), cex = 0.5, col = 2)
    points(test[[2]][, -3], pch = c("o"), cex = 0.5, col = 3)
    par(new = F)
}
```

```
plot.hyperplane(r2)
```

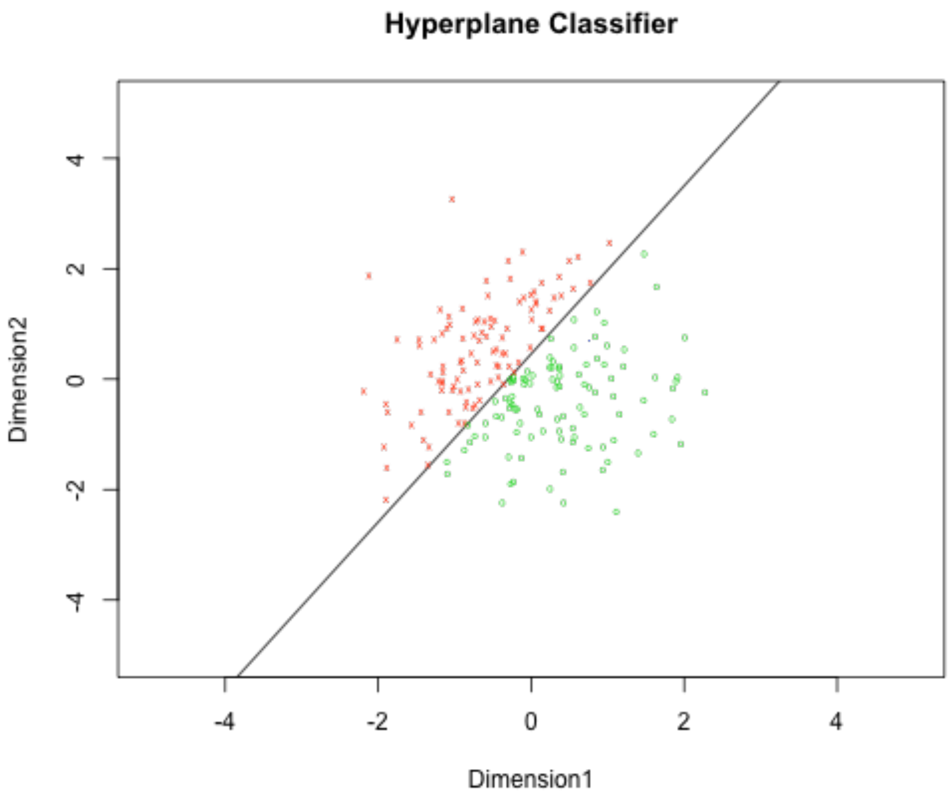


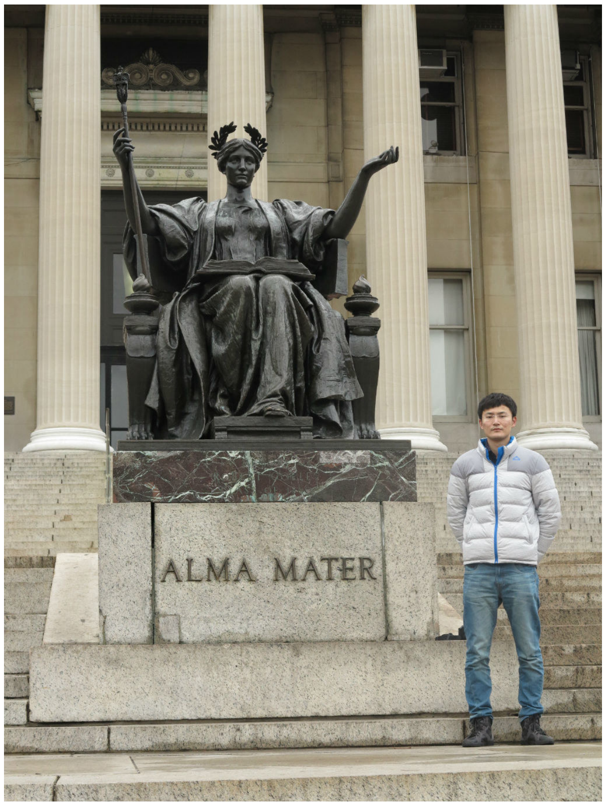
Then I will only plot the final affine hyperplane

Preparation for plotting hyperplane.

```
n <- nrow(r1)
r <- r1[n, ]
m <- (1/(sqrt(r[1]^2 + r[2]^2))) * r

R <- Null(m[1:2])
b <- R[2]/R[1]
z <- R + m[3] * c(m[1], m[2])
```

```
plot(z[2] ~ z[1], xlim = c(-5, 5), ylim = c(-5, 5), pch = 19,
cex = 0.001, xlab = "Dimension1",
    ylab = "Dimension2", main = "Hyperplane Classifier", col =
4)
abline(a = -sign(m[2]) * m[3] * sqrt(b^2 + 1), b = b, col = 1)
points(test[[1]][, -3], pch = c("x"), cex = 0.5, col = 2)
points(test[[2]][, -3], pch = c("o"), cex = 0.5, col = 3)
```




ALMA MATER