Problem Solutions

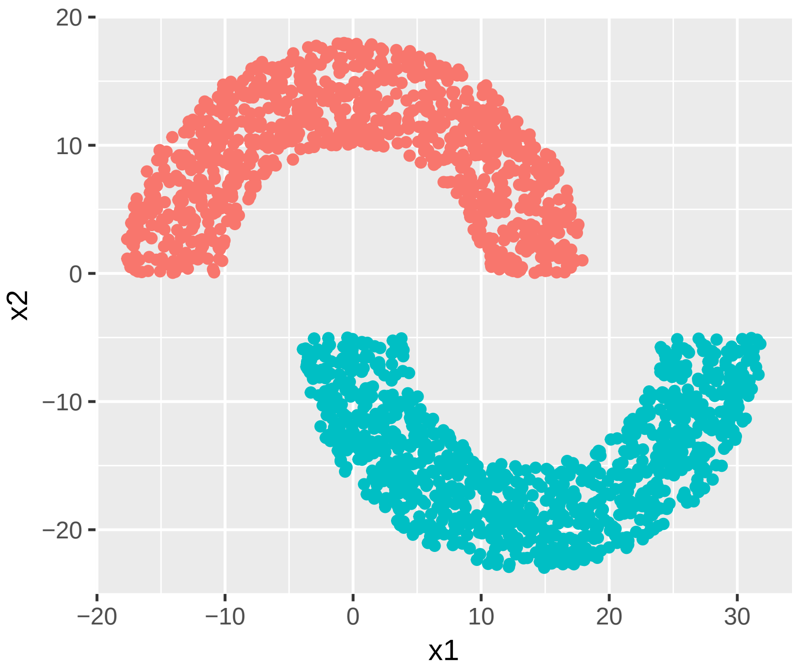
Chapter 3

Pierre Paquay

Problem 3.1

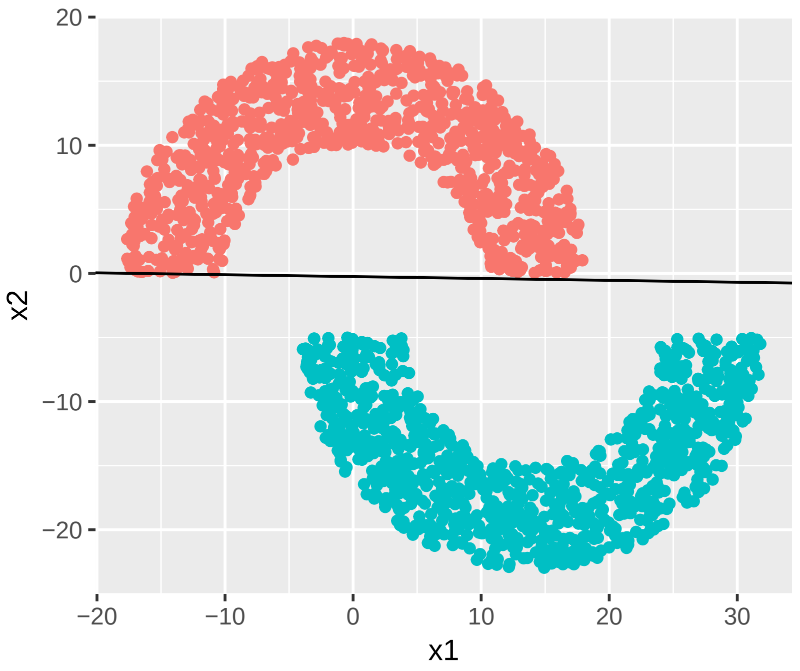
First, we generate 2000 examples uniformly for the two semi-circles, this means that we will have approximately 1000 examples for each class.

```r
set.seed(101)

init_data <- function(N, rad, thk, sep) {
  D <- data.frame(x = numeric(), y = numeric())
  y <- numeric()
  repeat {
    x1 <- runif(1, min = -25, max = 40)
    x2 <- runif(1, min = -30, max = 20)
    if ((x2 >= 0) && (rad^2 <= x1^2 + x2^2) && (x1^2 + x2^2 <= (rad + thk)^2)) {
      D <- rbind(D, c(x1, x2))
      y <- c(y, -1)
    }
    else if ((x2 < -sep) && (rad^2 <= (x1 - rad - thk / 2)^2 + (x2 + sep)^2) &&
             ((x1 - rad - thk / 2)^2 + (x2 + sep)^2 <= (rad + thk)^2)) {
      D <- rbind(D, c(x1, x2))
      y <- c(y, +1)
    }
    if (nrow(D) >= N)
      break
  }
  colnames(D) <- c("x1", "x2")

  return(cbind(D, y))
}

rad <- 10
thk <- 8
sep <- 5
D <- init_data(2000, rad, thk, sep)
p <- ggplot(D, aes(x = x1, y = x2, col = as.factor(y + 3))) + geom_point() +
  theme(legend.position = "none") +
  coord_fixed()
p
```



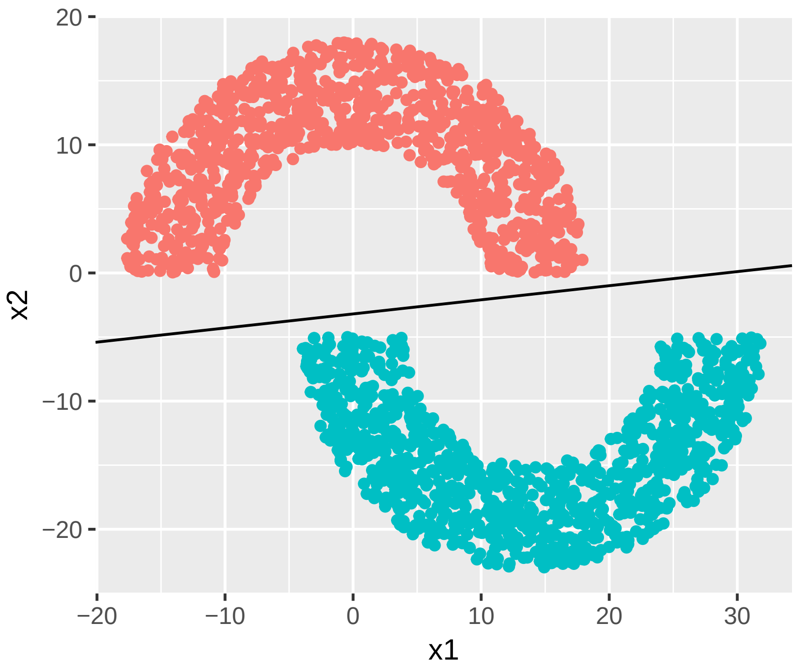
(*a*) Then, we run the PLA starting from $w = 0$ until it converges and we plot the data with the final hypothesis.

```r
h <- function(D, w) {
  scalar_prod <- cbind(1, D$x1, D$x2) %*% w

  return(as.vector(sign(scalar_prod)))
}

iter <- 0
w_PLA <- c(0, 0, 0)
repeat {
  y_pred <- h(D, w_PLA)
  D_mis <- subset(D, y != y_pred)
  if (nrow(D_mis) == 0)
    break
  xt <- D_mis[1, ]
  w_PLA <- w_PLA + c(1, xt$x1, xt$x2) * xt$y
  iter <- iter + 1
}
p + geom_abline(slope = -w_PLA[2] / w_PLA[3], intercept = -w_PLA[1] / w_PLA[3])
```



(b) Now, we use linear regression for classification to obtain w_{lin}.

```
X <- as.matrix(cbind(1, D[, c("x1", "x2")]))
y <- D$y
X_cross <- solve(t(X) %*% X) %*% t(X)
w_lin <- as.vector(X_cross %*% y)
p + geom_abline(slope = -w_lin[2] / w_lin[3], intercept = -w_lin[1] / w_lin[3])
```

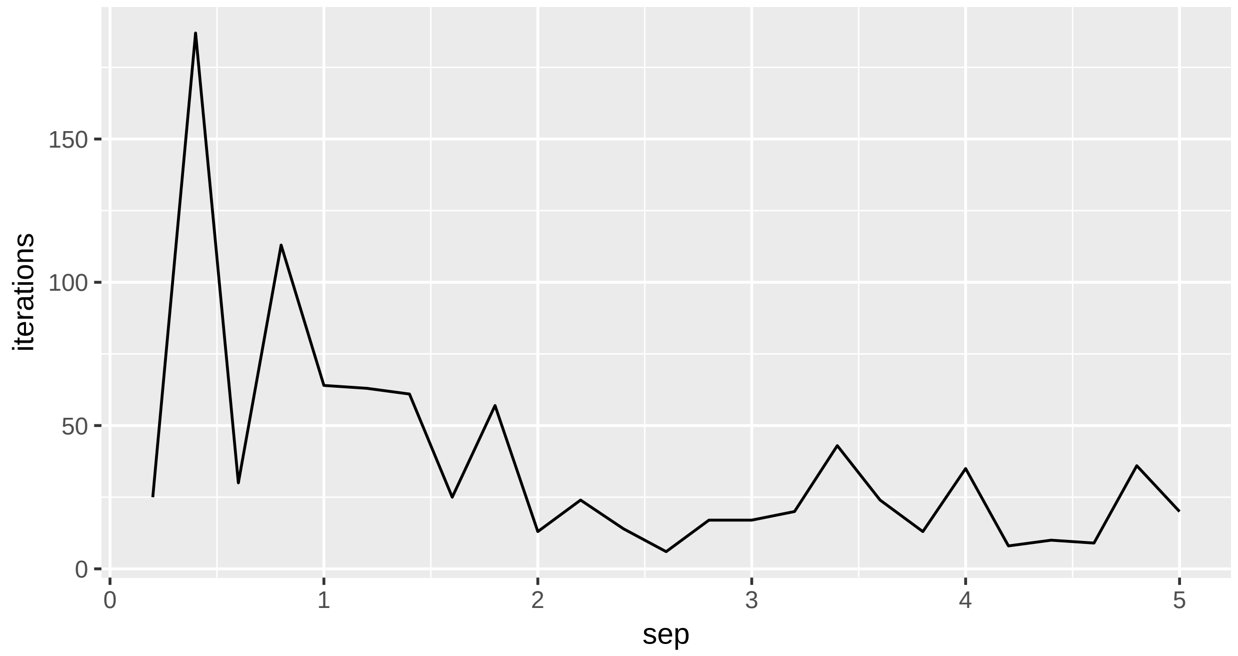


As we may see, linear regression can also be used for classification (the values $\text{sign}(w_{lin}^T x)$ will likely make good classification predictions). The linear regression weights w_{lin} are also an approximate solution for the perceptron model.

Problem 3.2

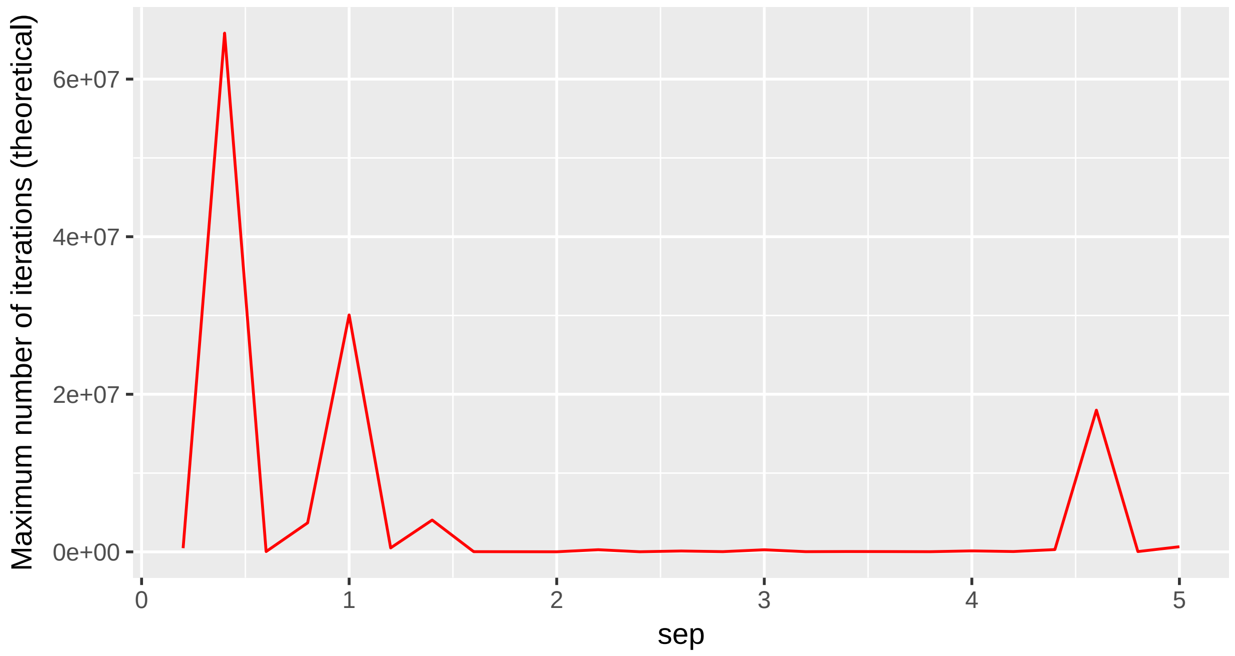
In this problem, we consider again the double-semi-circle of Problem 3.1 and we vary *sep* in the range $\{0.2, 0.4, \cdots, 5\}$, with these values we generate 2000 examples and we run PLA starting with $w = 0$. Below, we plot *sep* versus the number of iterations PLA takes to converge.

```r
set.seed(10)

sep_seq <- seq(0.2, 5, 0.2)
iterations <- numeric()
iterations_max <- numeric()
for (sep in sep_seq) {
  D <- init_data(2000, rad, thk, sep)
  iter <- 0
  w_PLA <- c(0, 0, 0)
  repeat {
    y_pred <- h(D, w_PLA)
    D_mis <- subset(D, y != y_pred)
    if (nrow(D_mis) == 0)
      break
    xt <- D_mis[1, ]
    w_PLA <- w_PLA + c(1, xt$x1, xt$x2) * xt$y
    iter <- iter + 1
  }
  iterations <- c(iterations, iter)
  R <- max(apply(cbind(1, D[, 1:2]), 1, FUN = function(x) sqrt(1 + x[2]^2 + x[3]^2)))
  Rho <- min(D$y * apply(cbind(1, D[, 1:2]), 1, FUN = function(x) sum(w_PLA * x)))
  iterations_max <- c(iterations_max, R^2 * sum(w_PLA^2) / Rho^2)
}
ggplot(data.frame(sep = sep_seq, iterations = iterations), aes(x = sep, y = iterations)) +
  geom_line()
```



We may see that the number of iterations tends to decrease when *sep* increases. This trend is confirmed by the theoretical results (see Problem 1.3), to see this we plot below *sep* versus the theoretical maximum number of iterations.

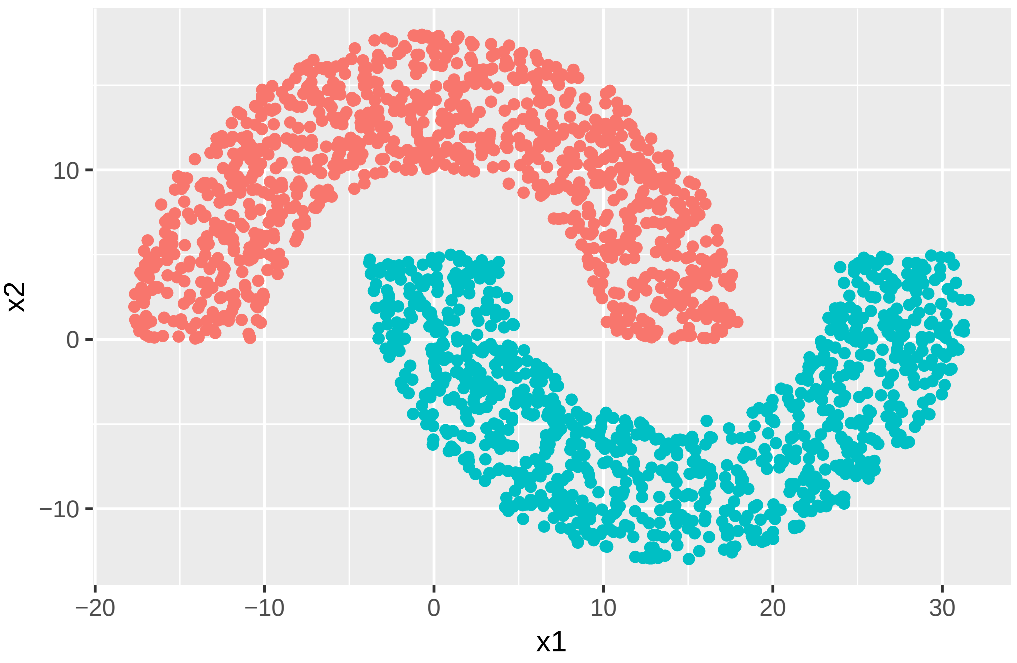
```r
ggplot(data.frame(sep = sep_seq, iterations = iterations_max), aes(x = sep, y = iterations)) +
  geom_line(col = "red") +
  ylab("Maximum number of iterations (theoretical)")
```



Problem 3.3

Here again, we consider the double-semi-circle of Problem 3.1 and we set $sep = -5$ (which makes the data non linearly separable) and we generate 2000 examples.

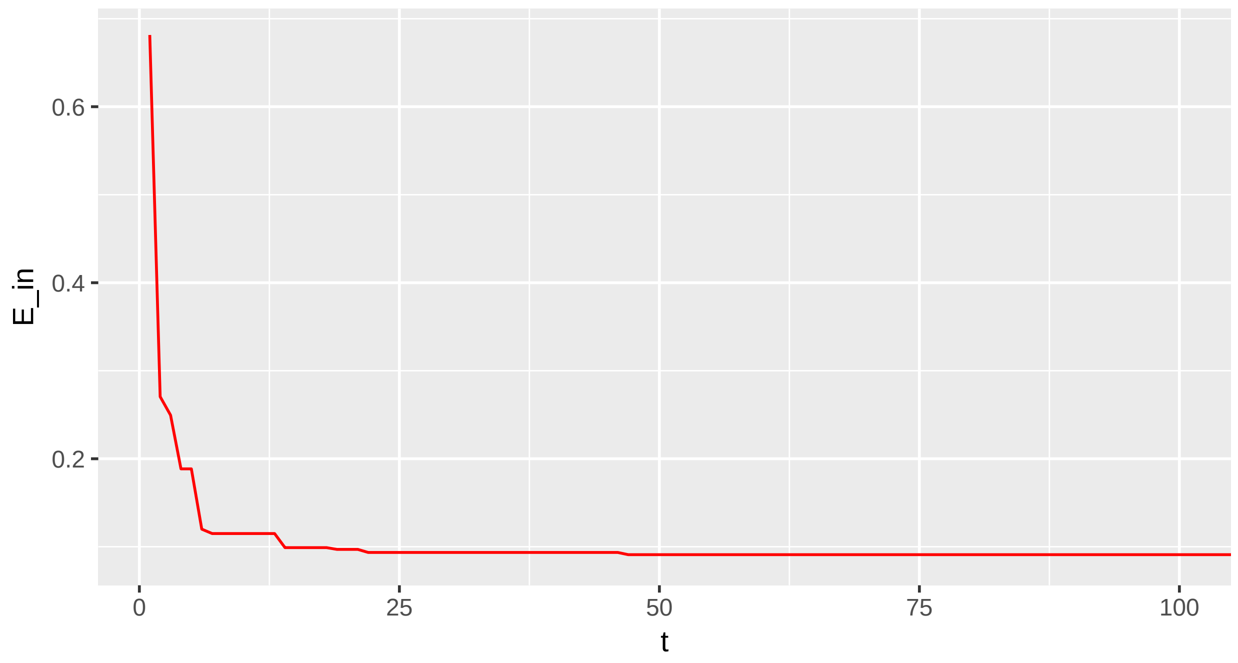
```r
set.seed(101)

sep <- -5
D <- init_data(2000, rad, thk, sep)
p <- ggplot(D, aes(x = x1, y = x2, col = as.factor(y + 3))) + geom_point() +
  theme(legend.position = "none") +
  coord_fixed()
p
```



(a) If we run PLA on these examples, it will never stop updating.

(b) Now, we run the pocket algorithm for 100000 iterations and we plot E_{in} versus the iteration number t for $t = 1, \cdots, 100$.

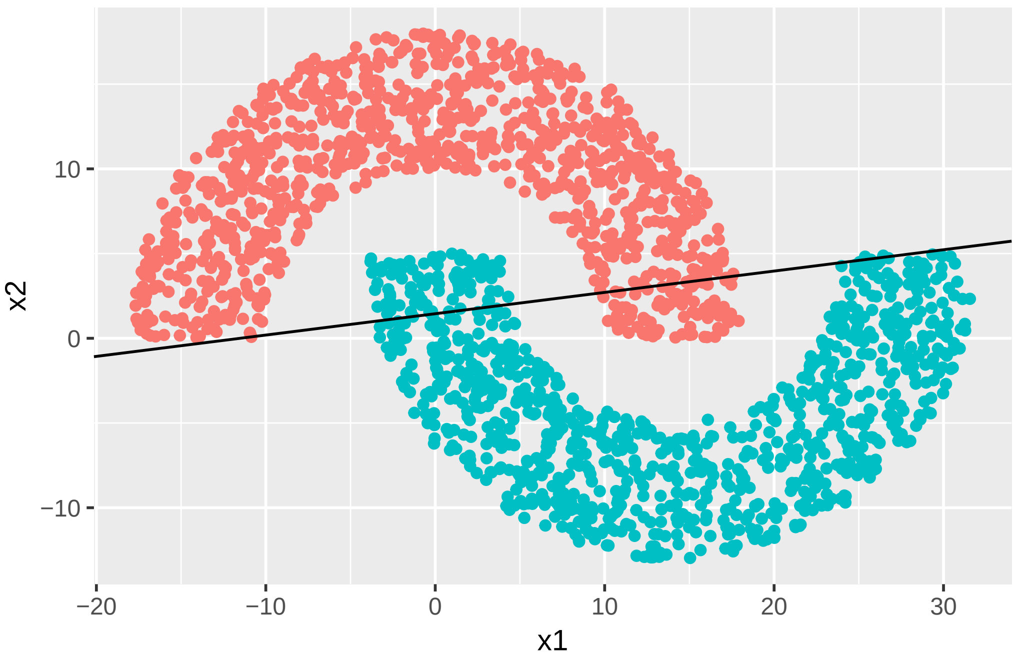
```
set.seed(101)

start_time_pocket <- Sys.time()
E_in <- numeric()
E_in_pocket <- numeric()
w <- c(0, 0, 0)
w_pocket <- w
E_in <- c(E_in, mean(D$y != h(D, w)))
E_in_pocket <- E_in
for (iter in 1:100000) {
  D_mis <- subset(D, y != h(D, w))
  if (nrow(D_mis) == 0)
    break
  xt <- D_mis[sample(nrow(D_mis), 1), ]
  w <- w + c(1, xt$x1, xt$x2) * xt$y
  E_in <- c(E_in, mean(D$y != h(D, w)))
  if (E_in[length(E_in)] < E_in_pocket[length(E_in_pocket)]) {
    w_pocket <- w
  }
  E_in_pocket <- c(E_in_pocket, mean(D$y != h(D, w_pocket)))
}
end_time_pocket <- Sys.time()
ggplot(data.frame(t = 1:100000, E_in = E_in_pocket[-1]), aes(x = t, y = E_in)) +
  geom_line(col = "red") +
  coord_cartesian(xlim = c(1, 100))
```



We clearly see that the E_{in} is monotonously decreasing (as opposed to what would happen if we had used the PLA).

(c) Below, we plot the data and the final hypothesis obtained in (b).

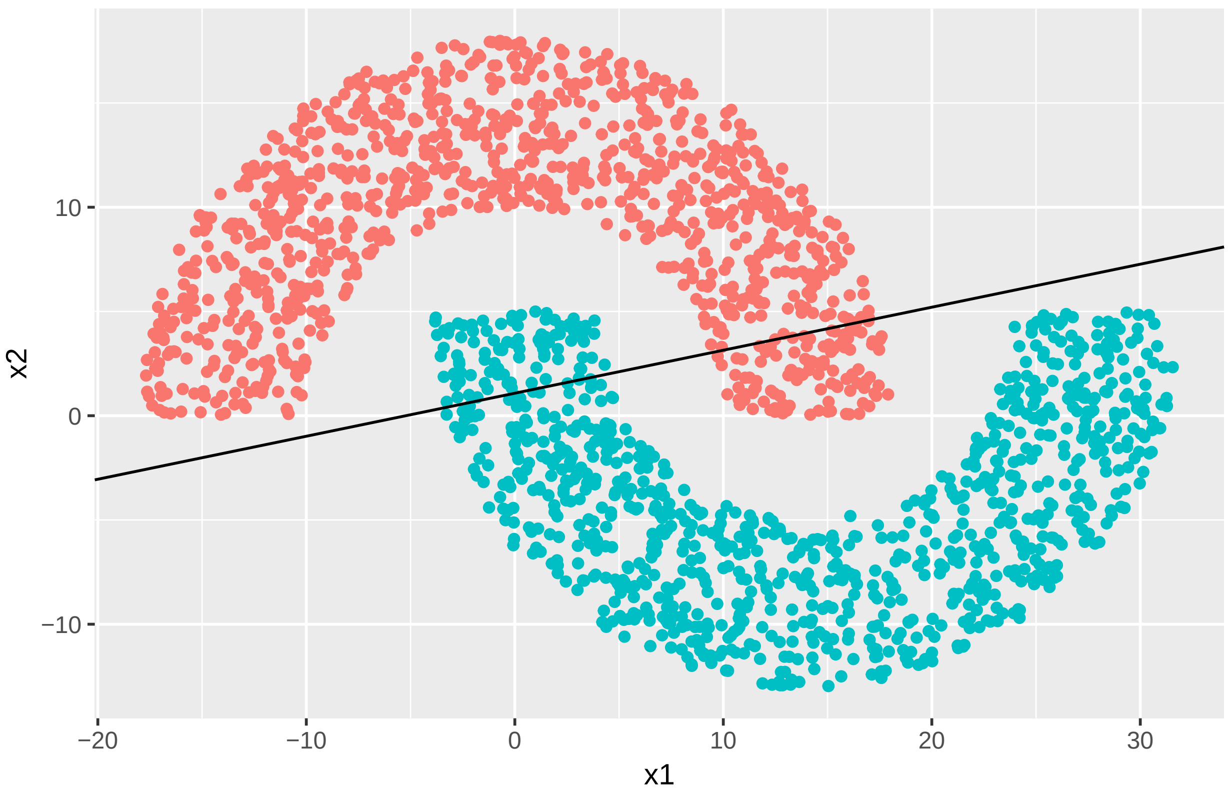
```
p + geom_abline(slope = -w_pocket[2] / w_pocket[3], intercept = -w_pocket[1] / w_pocket[3])
```



(d) Here, we use the linear regression algorithm to obtain w and we compare this result with the pocket algorithm in terms of computation time and quality of the solution.

```
start_time_lin <- Sys.time()
X <- as.matrix(cbind(1, D[, c("x1", "x2")]))
y <- D$y
X_cross <- solve(t(X) %*% X) %*% t(X)
```

```r
w_lin <- as.vector(X_cross %*% y)
E_in_lin <- mean(D$y != h(D, w_lin))
end_time_lin <- Sys.time()
p + geom_abline(slope = -w_lin[2] / w_lin[3], intercept = -w_lin[1] / w_lin[3])
```



For the pocket algorithm (with 100000 iterations), we have a computation time of 1.7378043, and for the linear regression algorithm, we have a computation time of 0.0048485. The linear regression algorithm is clearly better than the pocket algorithm in terms of computation time. When we take into account the quality of the solution, the pocket algorithm has a (final) E_{in} of 0.086, and the linear regression algorithm has a E_{in} of 0.0995. So, regarding the quality of the solution, the pocket algorithm is a little better than the linear regression algorithm.

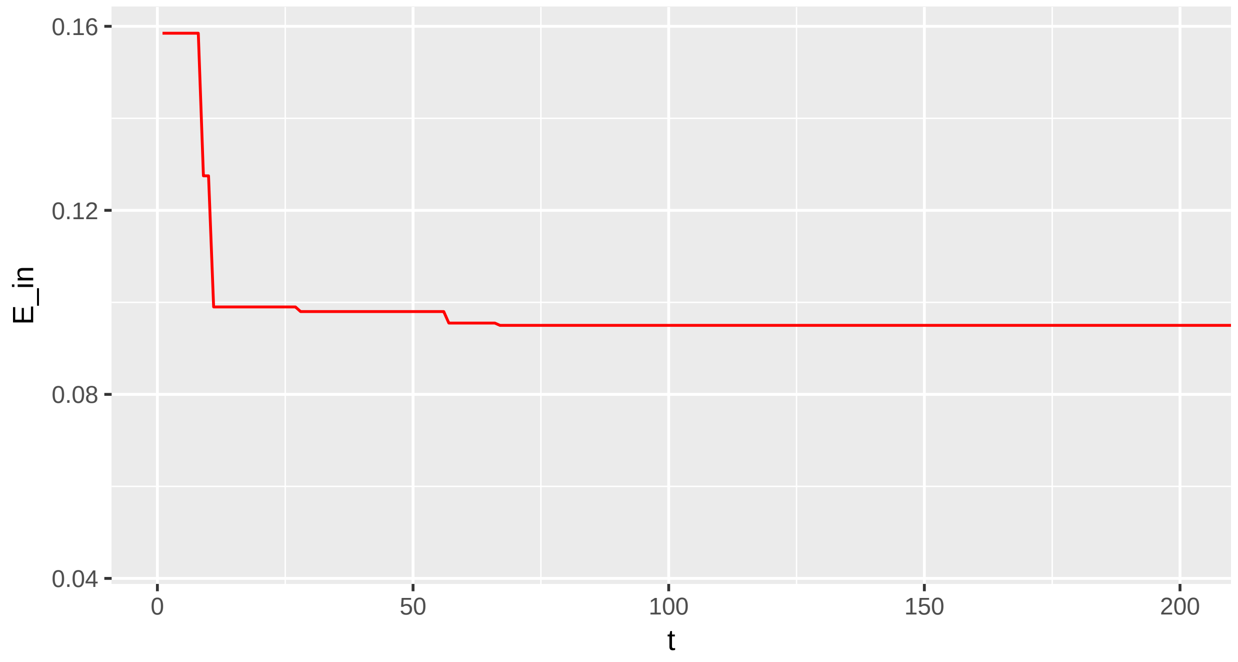
(e) Here, we repeat the points (b) to (d) with a 3rd order polynomial feature transform

$$\Phi(x) = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3).$$

First, we run the pocket algorithm for 100000 iterations and we plot E_{in} versus the iteration number t.

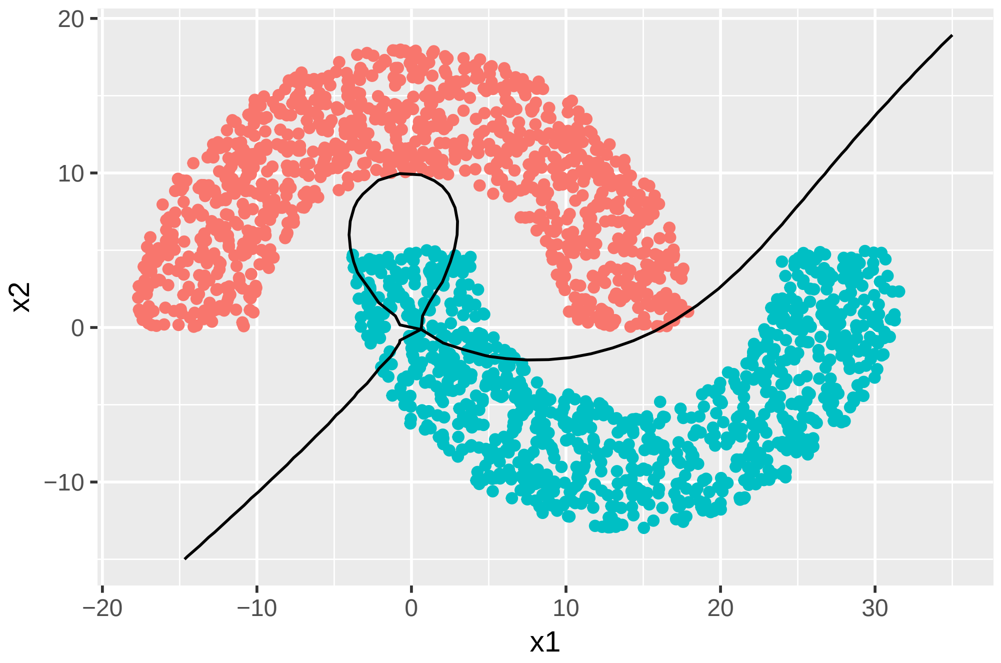
```r
set.seed(11)

D_trans <- data.frame(x1 = D$x1, x2 = D$x2,
                      x1_sq = D$x1^2, x1x2 = D$x1 * D$x2, x2_sq = D$x2^2,
                      x1_cub = D$x1^3, x1_sqx2 = D$x1^2 * D$ x2,
                      x1x2_sq = D$x1 * D$x2^2, x2_cub = D$x2^3, y = D$y)

h_trans <- function(D, w) {
  scalar_prod <- as.matrix(cbind(1, D[, 1:9])) %*% w

  return(as.vector(sign(scalar_prod)))
}
```

```r
start_time_pocket <- Sys.time()
E_in <- numeric()
E_in_pocket <- numeric()
w <- rep(0, 10)
w_pocket <- w
E_in <- c(E_in, mean(D_trans$y != h_trans(D_trans, w)))
E_in_pocket <- E_in
for (iter in 1:100000) {
  D_mis <- subset(D_trans, y != h_trans(D_trans, w))
  if (nrow(D_mis) == 0)
    break
  xt <- D_mis[sample(nrow(D_mis), 1), ]
  w <- w + c(1, as.numeric(xt[1:9])) * xt$y
  E_in <- c(E_in, mean(D_trans$y != h_trans(D_trans, w)))
  if (E_in[length(E_in)] < E_in_pocket[length(E_in_pocket)]) {
    w_pocket <- w
  }
  E_in_pocket <- c(E_in_pocket, mean(D_trans$y != h_trans(D_trans, w_pocket)))
}
end_time_pocket <- Sys.time()
ggplot(data.frame(t = 1:100000, E_in = E_in_pocket[-1]), aes(x = t, y = E_in)) +
  geom_line(col = "red") +
  coord_cartesian(xlim = c(1, 200))
```



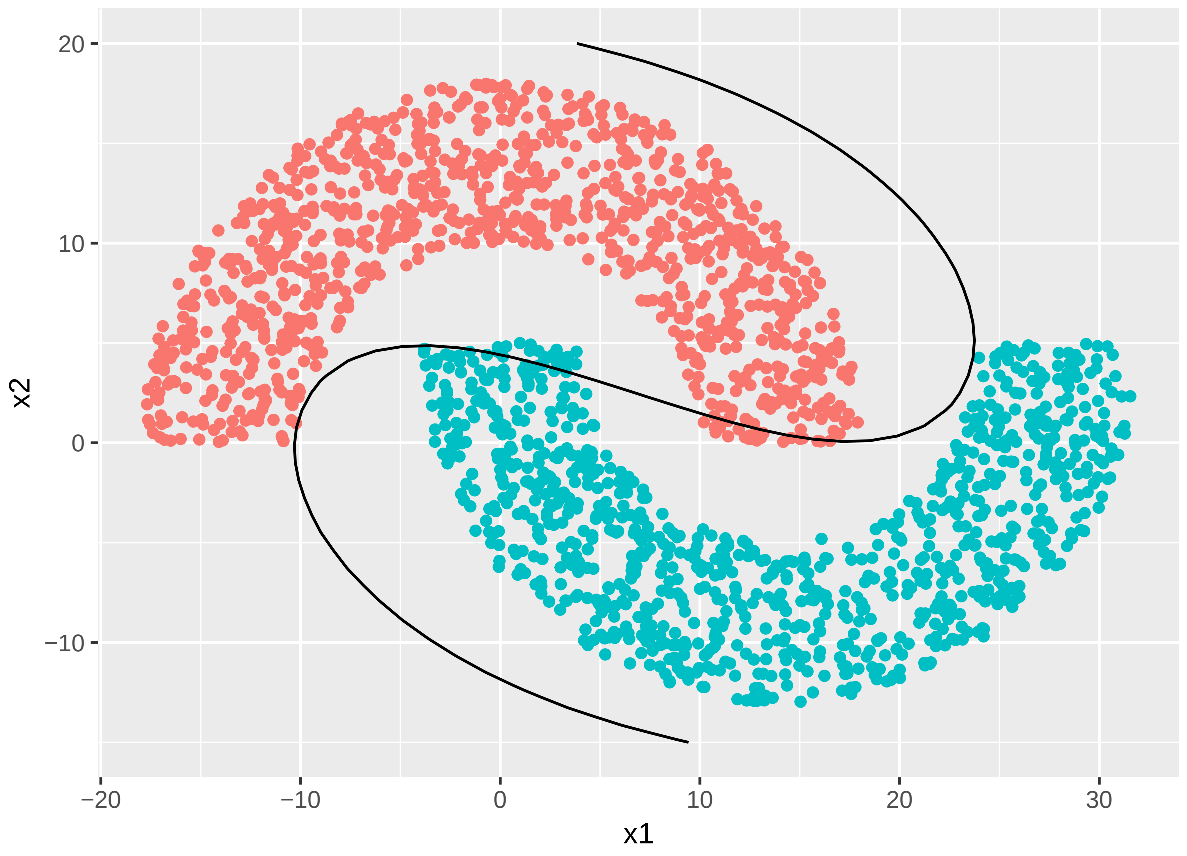
Then, we plot the data and the final hypothesis obtained above.

```r
cc <- emdbook::curve3d(1 * w_pocket[1] + x * w_pocket[2] + y * w_pocket[3] + x^2 * w_pocket[4] +
                         x * y * w_pocket[5] + y^2 * w_pocket[6] + x^3 * w_pocket[7] +
                         x^2 * y * w_pocket[8] + x * y^2 * w_pocket[9] + y^3 * w_pocket[10],
                       xlim = c(-20, 35), ylim = c(-15, 20), sys3d = "none")
dimnames(cc$z) <- list(cc$x, cc$y)
mm <- reshape2::melt(cc$z)
p + geom_contour(data = mm, aes(x = Var1, y = Var2, z = value), breaks = 0, colour = "black")
```



Finally, we use the linear regression algorithm to obtain the weights w.

```
start_time_lin <- Sys.time()
X <- as.matrix(cbind(1, D_trans[, 1:9]))
y <- D_trans$y
X_cross <- solve(t(X) %*% X) %*% t(X)
w_lin <- as.vector(X_cross %*% y)
E_in_lin <- mean(D_trans$y != h_trans(D_trans, w_lin))
end_time_lin <- Sys.time()
cc <- emdbook::curve3d(1 * w_lin[1] + x * w_lin[2] + y * w_lin[3] + x^2 * w_lin[4] +
                         x * y * w_lin[5] + y^2 * w_lin[6] + x^3 * w_lin[7] +
                         x^2 * y * w_lin[8] + x * y^2 * w_lin[9] + y^3 * w_lin[10],
                       xlim = c(-20, 35), ylim = c(-15, 20), sys3d = "none")
dimnames(cc$z) <- list(cc$x, cc$y)
mm <- reshape2::melt(cc$z)
p + geom_contour(data = mm, aes(x = Var1, y = Var2, z = value), breaks = 0, colour = "black")
```



For the pocket algorithm (with 100000 iterations), we have a computation time of 2.7625689, and for the linear regression algorithm, we have a computation time of 0.0066094. The linear regression algorithm is once again clearly better than the pocket algorithm in terms of computation time. When we take into account the quality of the solution, the pocket algorithm has a (final) E_{in} of 0.0445, and the linear regression algorithm has a E_{in} of 0.021. In this case, regarding the quality of the solution, the linear regression algorithm is also better than the pocket algorithm.

Problem 3.4

(a) We can write $e_n(w)$ as a piecewise function

$$e_n(w) = \left\{ \begin{array}{cl} 0 & \text{si} \quad y_n w^T x_n > 1 \\ (1 - y_n w^T x_n)^2 & \text{si} \quad y_n w^T x_n < 1 \end{array} \right. ;$$

and this function is actually continuous as we have

$$\lim_{w: y_n w^T x_n \to 1^{\pm}} e_n(w) = 0.$$

This function is also differentiable, to see this we note that

$$\nabla e_n(w) = \left\{ \begin{array}{cl} 0 & \text{si} \quad y_n w^T x_n > 1 \\ -2y_n(1 - y_n w^T x_n)x_n & \text{si} \quad y_n w^T x_n < 1 \end{array} \right. ,$$

and

$$\lim_{w: y_n w^T x_n \to 1^{\pm}} \nabla e_n(w) = 0.$$

(b) Let us consider first the case where $\text{sign}(w^T x_n) \neq y_n$, which means that $y_n w^T x_n \leq 0 < 1$. In this case, we have $[[\text{sign}(w^T x_n) \neq y_n]] = 1$ and $e_n(w) = (1 - y_n w^T x_n)^2 \geq 1$, consequently

$$[[\text{sign}(w^T x_n) \neq y_n]] \leq e_n(w).$$

Now we consider the second case where $\text{sign}(w^T x_n) = y_n$, which means that $y_n w^T x_n \geq 0$. In this case, we have $[[\text{sign}(w^T x_n) \neq y_n]] = 0$, $e_n(w) = (1 - y_n w^T x_n)^2 \geq 0$ if $0 \leq y_n w^T x_n < 1$ and $e_n(w) = 0$ if $y_n w^T x_n \geq 1$; consequently

$$[[\text{sign}(w^T x_n) \neq y_n]] \leq e_n(w).$$

In conclusion, we have that

$$E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} [[\text{sign}(w^T x_n) \neq y_n]] \leq \frac{1}{N} \sum_{n=1}^{N} e_n(w).$$

(c) If we apply SGD to our upper bound above, we get the following algorithm.

1. Select an initial w.
2. Repeat until CONDITION :
   Select (x_n, y_n) randomly and let $s_n = w^T x_n$. If $y_n s_n = y_n w^T x_n \leq 1$, we have

   $$\nabla e_n(w) = -2 y_n (1 - y_n s_n) x_n,$$

   and we update w as

   $$w \leftarrow w - \eta \nabla e_n(w) = w + 2\eta y_n (1 - y_n s_n) x_n = w + 2\eta (y_n - s_n) x_n = w + \eta' (y_n - s_n) x_n.$$

   And if $y_n s_n = y_n w^T x_n > 1$, we have $\nabla e_n(w) = 0$, and we update w as

   $$w \leftarrow w - \eta \nabla e_n(w) = w - \eta \cdot 0 = w.$$

Which is exactly the Adaline algorithm.

Problem 3.5

(a) We can write $e_n(w)$ as a piecewise function

$$e_n(w) = \left\{ \begin{array}{ccc} 0 & \text{si} & y_n w^T x_n > 1 \\ 1 - y_n w^T x_n & \text{si} & y_n w^T x_n < 1 \end{array} \right. ;$$

and this function is actually continuous everywhere as we have

$$\lim_{w : y_n w^T x_n \to 1^{\pm}} e_n(w) = 0.$$

However, this function is not differentiable everywhere, to see this we note that

$$\nabla e_n(w) = \left\{ \begin{array}{ccc} 0 & \text{si} & y_n w^T x_n > 1 \\ -y_n x_n & \text{si} & y_n w^T x_n < 1 \end{array} \right. ;$$

moreover

$$\lim_{w : y_n w^T x_n \to 1^{+}} \nabla e_n(w) = 0 \text{ and } \lim_{w : y_n w^T x_n \to 1^{-}} \nabla e_n(w) = -y_n x_n \neq 0.$$

Thus, the function $e_n(w)$ is differentiable everywhere except when $y_n w^T x_n = 1$ ($\Leftrightarrow y_n = w^T x_n$).

(b) Let us consider first the case where $\text{sign}(w^Tx_n) \neq y_n$, which means that $y_n w^T x_n \leq 0 < 1$. In this case, we have $[[\text{sign}(w^Tx_n) \neq y_n]] = 1$ and $e_n(w) = 1 - y_n w^T x_n \geq 1$, consequently

$$[[\text{sign}(w^Tx_n) \neq y_n]] \leq e_n(w).$$

Now we consider the second case where $\text{sign}(w^Tx_n) = y_n$, which means that $y_n w^T x_n \geq 0$. In this case, we have $[[\text{sign}(w^Tx_n) \neq y_n]] = 0$, $e_n(w) = 1 - y_n w^T x_n \geq 0$ if $0 \leq y_n w^T x_n < 1$ and $e_n(w) = 0$ if $y_n w^T x_n \geq 1$; consequently

$$[[\text{sign}(w^Tx_n) \neq y_n]] \leq e_n(w).$$

In conclusion, we have that

$$E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N}[[\text{sign}(w^Tx_n) \neq y_n]] \leq \frac{1}{N}\sum_{n=1}^{N} e_n(w).$$

(c) If we apply SGD to our upper bound above, we get the following algorithm.

1. Select an initial w.
2. Repeat until CONDITION :
   Select (x_n, y_n) randomly. If $y_n w^T x_n < 1$, we have
   $$\nabla e_n(w) = -y_n x_n,$$
   and we update w as
   $$w \leftarrow w - \eta \nabla e_n(w) = w + \eta y_n x_n.$$
   And if $y_n s_n = y_n w^T x_n > 1$, we have $\nabla e_n(w) = 0$, and we update w as
   $$w \leftarrow w - \eta \nabla e_n(w) = w - \eta \cdot 0 = w.$$

Which is a new perceptron learning algorithm.

Problem 3.6

(a) For linearly separable data, we obviously have that there exists w^* so that $y_n(w^*)^T x_n > 0$ for all $n = 1, \cdots, N$. By the density property of the real numbers, we know that we may find $\epsilon > 0$ so that $y_n(w^*)^T x_n \geq \epsilon$ for all $n = 1, \cdots, N$, and consequently if we let $w = w^*/\epsilon$, we get

$$y_n w^T x_n \geq 1$$

for all $n = 1, \cdots, N$.

(b) The task of finding w for separable data may be formulated as the following linear program

$$\begin{cases} \min_w & c^T w \\ \text{subject to} & Aw \leq b \end{cases}$$

where $c^T = (0, \cdots, 0)$, $b^T = (-1, \cdots, -1)$, and

$$A = -\begin{pmatrix} - & y_1 x_1^T & - \\ \vdots & \vdots & \vdots \\ - & y_N x_N^T & - \end{pmatrix} (N \times (d+1)).$$

(c) When the data is not separable, the minimization problem may be formulated as a linear program as follows

$$\begin{cases} \min_{(w,\xi)} & c^T (w,\xi)^T \\ \text{subject to} & A(w,\xi)^T \leq b \end{cases}$$

where $c^T = (0, \cdots, 0, 1, \cdots, 1)$, $b^T = (-1, \cdots, -1, 0, \cdots, 0)$, and

$$A = -\left(\begin{array}{ccc|ccc} - & y_1 x_1^T & - & 1 & & \\ \vdots & \vdots & \vdots & & \ddots & \\ - & y_N x_N^T & - & & & 1 \\ \hline 0 & \cdots & 0 & 1 & & \\ \vdots & \vdots & \vdots & & \ddots & \\ 0 & \cdots & 0 & & & 1 \end{array}\right) (2N \times (d+1+N)).$$

(d) In Problem 3.5, we sought to minimize the expression

$$\frac{1}{N}\sum_{n=1}^{N} e_n(w)$$

with

$$e_n(w) = \begin{cases} 0 & \text{si} \quad y_n w^T x_n > 1 \\ 1 - y_n w^T x_n & \text{si} \quad y_n w^T x_n < 1 \end{cases}.$$

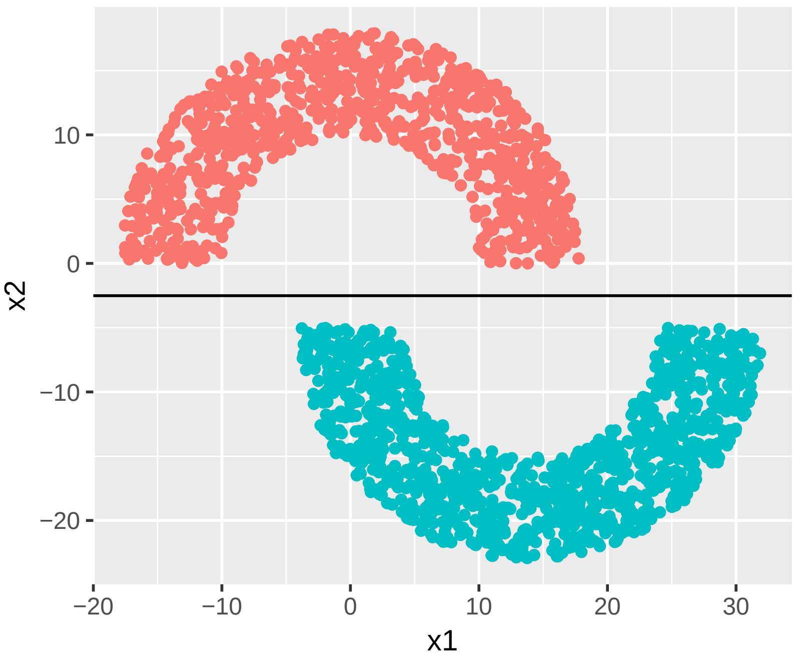
If we take a look at $e_n(w)$, we may note that when x_n is correctly classified by w and at least at a margin of one of the linear separator ($y_n w^T x_n \geq 1$), we get $e_n(w) = 0$ which means that in this case this term does not contribute to the overall error. However, when x_n is in the margin of one, the deeper x_n is into the margin of one, the higher the term $e_n(w) = 1 - y_n w^T x_n$ contributes to the overall error. For example, if x_n is correctly classified by w but into the margin of one ($0 < y_n w^T x_n < 1$), then the error term for this point is $0 < e_n(w) < 1$; and if x_n is not correctly classified by w ($y_n w^T x_n < 0$), then the error term for this point is $e_n(w) > 1$. In conclusion, the overall error characterizes the amount of violation of the margin, which is exactly what we seek to minimize in point (c) above.

Problem 3.7

First, we use the linear programming algorithm from Problem 3.6 on the learning task in Problem 3.1 for the separable case.

```r
set.seed(10)

rad <- 10
thk <- 8
sep <- 5
D <- init_data(2000, rad, thk, sep)

p <- ggplot(D, aes(x = x1, y = x2, col = as.factor(y + 3))) + geom_point() +
  theme(legend.position = "none") +
  coord_fixed()

d <- 2
N <- nrow(D)
c_T <- rep(0, d + 1)
b_T <- rep(-1, N)
A <- -diag(D$y) %*% as.matrix(cbind(1, D[, 1:2]))
dir <- rep("<=", N)
linear_prog <- lp("min", c(c_T, -c_T), cbind(A, -A), const.dir = dir, const.rhs = b_T)
w <- linear_prog$solution[1:(d+1)] - linear_prog$solution[(d+2):(2 * (d + 1))]
```

```r
X <- as.matrix(cbind(1, D[, c("x1", "x2")]))
y <- D$y
X_cross <- solve(t(X) %*% X) %*% t(X)
w_lin <- as.vector(X_cross %*% y)

D_trans <- data.frame(x1 = D$x1, x2 = D$x2,
                      x1_sq = D$x1^2, x1x2 = D$x1 * D$x2, x2_sq = D$x2^2,
                      x1_cub = D$x1^3, x1_sqx2 = D$x1^2 * D$ x2,
                      x1x2_sq = D$x1 * D$x2^2, x2_cub = D$x2^3, y = D$y)
X <- as.matrix(cbind(1, D_trans[, 1:9]))
y <- D_trans$y
X_cross <- solve(t(X) %*% X) %*% t(X)
w_pol <- as.vector(X_cross %*% y)

p + geom_abline(slope = -w[2] / w[3], intercept = -w[1] / w[3])
```

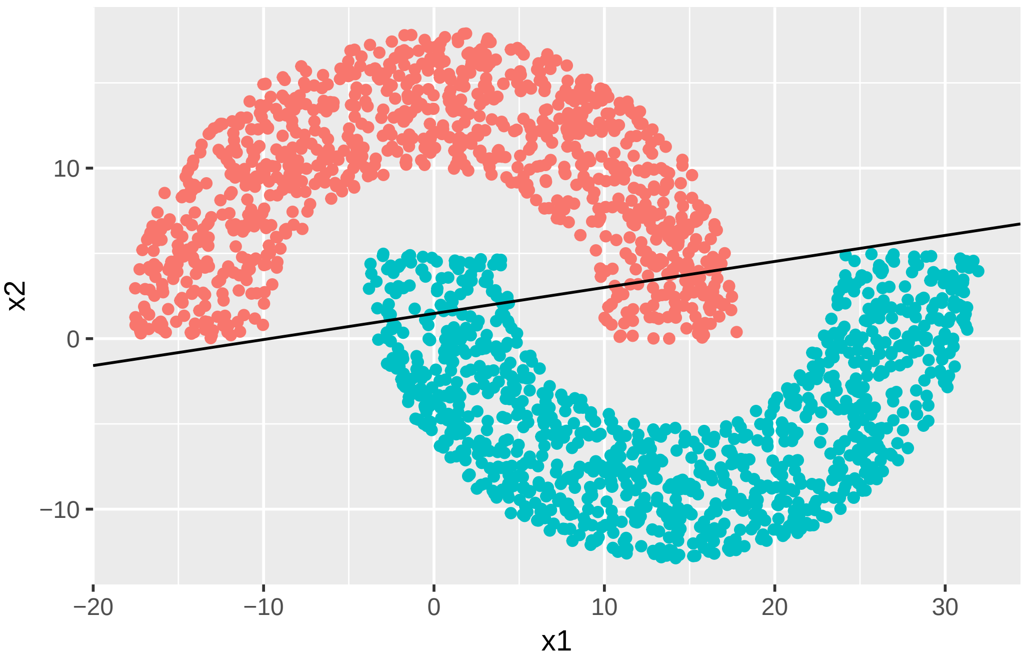


As we may see, our linear programming algorithm solution perfectly separates the dataset so we have an E_{in} of 0; the linear regression approach gives us an E_{in} of 0, and the 3rd order polynomial feature transform gives us an E_{in} of 0 as well.

Now, we use the linear programming algorithm from Problem 3.6 on the learning task in Problem 3.1 for the non separable case.

```r
set.seed(10)

rad <- 10
thk <- 8
sep <- -5
D <- init_data(2000, rad, thk, sep)

p <- ggplot(D, aes(x = x1, y = x2, col = as.factor(y + 3))) + geom_point() +
  theme(legend.position = "none") +
  coord_fixed()
```

```
d <- 2
N <- nrow(D)
c_T <- c(rep(0, d + 1), rep(1, N))
b_T <- c(rep(-1, N), rep(0, N))
A1 <- rbind(diag(D$y) %*% as.matrix(cbind(1, D[, 1:2])), matrix(0, nrow = N, ncol = d + 1))
A2 <- rbind(diag(1, nrow = N, ncol = N), diag(1, nrow = N, ncol = N))
A <- -cbind(A1, A2)
dir <- rep("<=", 2 * N)
linear_prog <- lp("min", c(c_T, -c_T), cbind(A, -A), const.dir = dir, const.rhs = b_T)
w <- linear_prog$solution[1:(d + 1 + N)] - linear_prog$solution[(d + 1 + N + 1):(2 * (d + 1 + N))]

X <- as.matrix(cbind(1, D[, c("x1", "x2")]))
y <- D$y
X_cross <- solve(t(X) %*% X) %*% t(X)
w_lin <- as.vector(X_cross %*% y)

D_trans <- data.frame(x1 = D$x1, x2 = D$x2,
                      x1_sq = D$x1^2, x1x2 = D$x1 * D$x2, x2_sq = D$x2^2,
                      x1_cub = D$x1^3, x1_sqx2 = D$x1^2 * D$ x2,
                      x1x2_sq = D$x1 * D$x2^2, x2_cub = D$x2^3, y = D$y)
X <- as.matrix(cbind(1, D_trans[, 1:9]))
y <- D_trans$y
X_cross <- solve(t(X) %*% X) %*% t(X)
w_pol <- as.vector(X_cross %*% y)

p + geom_abline(slope = -w[2] / w[3], intercept = -w[1] / w[3])
```



Here, our linear programming algorithm solution gives us an E_{in} of 0.072; the linear regression approach gives us an E_{in} of 0.0855, and the 3rd order polynomial feature transform gives us an E_{in} of 0.0205 which is the best of the three approaches.