Inteligência Computacional I - Lista II

Luan Vieira

```r
# Instalando a única biblioteca externa utilizada, caso necessário
if (!requireNamespace("ggplot2"))
  install.packages('ggplot2')
# Importando a biblioteca
library("ggplot2")
```

Questão 3

Seja f^* a versão ruidosa da função f. Temos que:

$$P[h(x) \neq f(x)] = P\left[(h(x) \neq f^*(x)) \cap (f^*(x) = f(x))\right] + P\left[(h(x) = f(x)) \cap (f^*(x) \neq f(x))\right]$$

Consideraremos que a função h aproxima f^* e f^* aproxima f de maneira independente.

Então $P\left[(h \text{ aproxima } f^*) \cap (f^* \text{ aproxima } f)\right] = P\left[h \text{ aproxima } f^*\right] \cdot P\left[f^* \text{ aproxima } f\right]$

$$\begin{aligned}
&P\left[(h(x) \neq f^*(x)) \cap (f^*(x) = f(x))\right] + P\left[(h(x) = f(x)) \cap (f^*(x) \neq f(x))\right] = \\
&P\left[h(x) \neq f^*(x)\right] \cdot P\left[f^*(x) = f(x)\right] + P\left[(h(x) = f(x)\right] \cdot P\left[f^*(x) \neq f(x)\right] = \\
&\mu \cdot \lambda + (1-\mu) \cdot (1-\lambda)
\end{aligned}$$

Resposta: letra e

Questão 4

$$\begin{aligned}
&P\left[h(x) \neq f^*(x)\right] = \\
&\mu \cdot \lambda + (1-\mu) \cdot (1-\lambda) = \\
&\mu \cdot \lambda + 1 - \lambda - \mu + \mu \cdot \lambda = \\
&(2 \cdot \mu \cdot \lambda - \mu) + (1-\lambda) = \\
&\mu \cdot (2 \cdot \lambda - 1) + (1-\lambda) \\
&\text{Não depende de } \mu \iff 2 \cdot \lambda - 1 = 0 \iff \lambda = \frac{1}{2}
\end{aligned}$$

Resposta: letra b

Questões 5 a 8 - Ilustrando o problema

Como estamos trabalhando em duas dimensões, dados x e x_2 dois pontos quaisquer em $\mathbb{R}^2 \cap [-1, 1]$, uma reta do tipo $y = ax + b$ é definida univocamente por eles.

```r
# gerar função f
gerar_f <- function() {
  x1 <- runif(2, -1, 1)
```

```r
  x2 <- runif(2, -1, 1)
# calcular a e b, coeficiente angular e intercepto, respectivamente
# da reta (f) passando por x1 e x2
  a <- (x2[2] - x1[2]) / (x2[1] - x1[1])
  b <- x1[2] - a * x1[1]
  return(list(x1 = x1, x2 = x2, a=a, b=b))
}
```

```r
set.seed(1)
f <- gerar_f()
print("Pontos que geraram f e coeficiente angular e intercepto da reta")
```

```
## [1] "Pontos que geraram f e coeficiente angular e intercepto da reta"
```

```r
print(f)
```

```
## $x1
## [1] -0.4689827 -0.2557522
##
## $x2
## [1] 0.1457067 0.8164156
##
## $a
## [1] 1.744243
##
## $b
## [1] 0.5622676
```

```r
gerar_dados <- function(N) {
    X <- matrix(runif(2*N, -1, 1), ncol = 2)
    colnames(X) <- c("x1", "x2")
    rownames(X) <- paste0("p", 1:N)
    return(X)
}
```

```r
avaliar_dados <- function(X, avaliar_intercepto, avaliar_coef_angular) {
  sinal <- ifelse(X[, 2] - avaliar_coef_angular * X[, 1]
                  - avaliar_intercepto > 0, 1, -1)
  return(sinal)
}
```

```r
set.seed(1)
X <- gerar_dados(100)
sinal <- avaliar_dados(X,avaliar_coef_angular = f$a, avaliar_intercepto = f$b)
print("Visualizando primeiras observações:")
```

```
## [1] "Visualizando primeiras observações:"
```

```r
print(head(cbind(X,sinal)))
```

```
##            x1         x2 sinal
## p1 -0.4689827  0.3094479     1
## p2 -0.2557522 -0.2936055    -1
## p3  0.1457067 -0.4594797    -1
## p4  0.8164156  0.9853681    -1
## p5 -0.5966361  0.2669865     1
## p6  0.7967794 -0.5735837    -1
```

Até aqui utilizamos os mesmos conceitos da lista 1. No entanto, agora iremos utilizar a regressão linear ao invés do algoritmo perceptron para nosso problema de classificação binária.

```
reg <- lm(sinal ~ X)
reg$coefficients
```

```
## (Intercept)         Xx1         Xx2
##  -0.3078366  -1.3774365   0.4865835
```

```
intercepto <- -reg$coefficients[1]/reg$coefficients[3]
coef_angular <- -reg$coefficients[2]/reg$coefficients[3]

intercepto
```

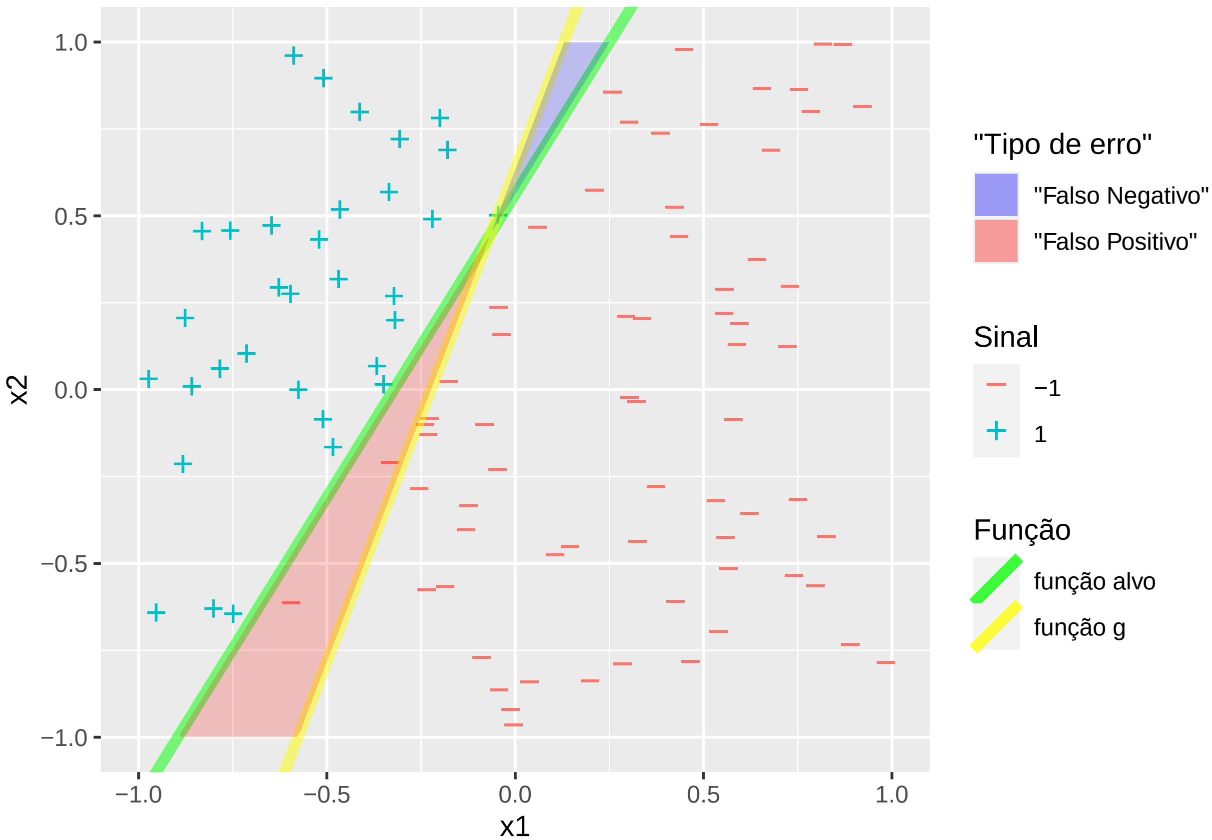
```
## (Intercept)
##    0.632649
```

```
coef_angular
```

```
##      Xx1
## 2.830833
```

```
# Visualizar pontos e função alvo
library(ggplot2)

# Criar dadoframe com dados e sinal
df <- data.frame(X, sinal)

# Criar dataframe auxiliar para visualização das áreas entre as retas
dado_auxiliar <- data.frame(x = seq(-1, 1, 0.01))
dado_auxiliar$f <- sapply(dado_auxiliar$x,
                          FUN = function(x) {f$b + f$a * x})
dado_auxiliar$g <- sapply(dado_auxiliar$x,
                          FUN = function(x) {intercepto + coef_angular * x})

# Visualizar problema
ggplot(df, aes(x=X[,1], y=X[,2], size = 3,
               color=factor(sinal), shape = factor(sinal))) +
  geom_point(show.legend = TRUE) +
  xlim(-1, 1) + ylim(-1, 1) +
  geom_abline(aes(slope = f$a, intercept = f$b, linetype = "função alvo"),
    color = "green", linewidth = 2, alpha = 0.5) +
  geom_abline(aes(slope = coef_angular, intercept = intercepto,
    linetype = "função g"), color = "yellow", linewidth = 2, alpha = 0.5) +
  scale_shape_manual(values = c("-", "+"), name = "Sinal") +
  scale_linetype_manual(values = c("solid", "solid"), name = "Função") +
  scale_fill_manual(values = c("blue", "red"),
    labels = c('"Falso Negativo"', '"Falso Positivo"'),
    name = '"Tipo de erro"') +
  geom_ribbon(data = subset(dado_auxiliar, f > g),
    aes(x = x, ymin = pmax(g, -1), ymax = f, fill = '"Falso Positivo"'),
    alpha = 0.2, inherit.aes = FALSE) +
  geom_ribbon(data = subset(dado_auxiliar, f < g),
    aes(x = x, ymin = f, ymax = pmin(g,1), fill = '"Falso Negativo"'),
    alpha = 0.2, inherit.aes = FALSE) +
  guides(color = guide_legend(override.aes =
                    list(shape = c("-", "+"), size = 5))) +
```

```r
guides(linetype = guide_legend(override.aes =
                    list(color = c("green", "yellow"), shape = c(NA, NA)))) +
guides(fill = guide_legend(override.aes = list(shape = c(NA,NA)))) +
labs(color = "Sinal", shape = "Sinal", linetype = "Função",
     fill = '"Tipo de erro"', x = "x1", y = "x2") +
scale_size(guide = FALSE) +
theme_gray()
```



Neste exemplo, usando uma amostra de tamanho 100 para treinamento, conseguimos ver como a função encontrada pelo método de regressão linear aproxima a função alvo. No entanto, o modelo linear não tem como propósito separar todos os dados. Por consequência, a existência de erros neste experimento é esperada, especialmente com um número de observações grande.

Quando a função alvo é maior que a função encontrada pelo modelo linear, o modelo classificaria os dados entre g e f como +1, quando na verdade deveriam ser -1. Esse erro é chamado no gráfico de "Falso Positivo". As aspas são utilizadas para evidenciar que não se trata do uso comum de Falso Positivo e Falso Negativo usados em aprendizado de máquina, quando os dados são classificados de acordo com a ocorrência ou não de algum fator de interesse.

Similarmente, quando a reta encontrada pelo modelo de regressão linear fica acima da função alvo, temos pontos classificados como "Falso Negativo" na área entre f e g, pois o modelo os classificará como -1, quando na verdade são +1.

Questão 5

```r
# Inicializar vetores
prob_erro_dentro <- c()
f_coef_angular <- c()
f_intercepto <- c()
# Número de experimentos
num_experimentos <- 1000
# Tamanho da amostra (treino)
N = 100
# Inicializar matriz que receberá os pe pesos de regressão a cada experimento
reg <- matrix(data = NA, nrow = num_experimentos,ncol =3)

# Para cada experimento
for (i in 1:num_experimentos){
  # Gerar f e armazenar intercepto e coef angular
  f <- gerar_f()
  f_intercepto[i] <- f$b
  f_coef_angular[i] <- f$a
  # Gerar dados
  X <- gerar_dados(N)
  # Avaliar dados com função alvo gerada
  sinal_in <- avaliar_dados(X, avaliar_coef_angular = f_coef_angular[i],
                            avaliar_intercepto = f_intercepto[i])
  # Obter coeficientes de regressão
  reg[i,] <- lm(sinal_in ~ X[,1] + X[,2])$coefficients
  # Prever sinais nos dados de treino
  sinal_pred_in <- sign(cbind(1,X) %*% reg[i,])
  # Estimar erro dentro da amostra no experimento
  prob_erro_dentro[i] = length(which(sinal_in != sinal_pred_in))/N
}
# Mostrar informações sobre o erro nos experimentos
summary(prob_erro_dentro)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.02000 0.03000 0.03757 0.05000 0.15000
```

A média dos erros, de aproximadamente 3,8%, está mais próxima da alternativa c.

Resposta: letra c

Questão 6

```r
# Fixar seed para reprodutibilidade
set.seed(1)
# Número de experimentos
num_experimentos <- 1000
# Definir tamanho da amostra de teste
N_out <- 1000
# Inicializar vetor de erro
prob_erro_fora <- c()

# Para cada experimento
for (i in 1:num_experimentos){
```

```r
  # Gerar dados de teste
  X_out <- gerar_dados(N_out)
  # Checar sinal verdadeiro, de acordo com a função alvo
  sinal_out <- avaliar_dados(X_out, avaliar_coef_angular = f_coef_angular[i],
                             avaliar_intercepto = f_intercepto[i])
  # Prever sinal de acordo com a regressão treinada no experimento
  sinal_pred_out <- sign(cbind(1,X_out) %*% reg[i,])
  # Estimar probabilidade de erro fora da amostra no experimento
  prob_erro_fora[i] = length(which(sinal_out != sinal_pred_out))/N_out
}
# Estimar erro fora da amostra no experimento
summary(prob_erro_fora)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.02500 0.04100 0.04601 0.06025 0.19600
```

Conseguimos observar que o erro fora da amostra E_{out} aparenta acompanhar bem o erro dentro da amostra E_{in}. Ainda, $E_{out} > E_{in}$, embora não seja uma regra, é também um resultado esperado.

Podemos comparar este resultado com o da questão 10 da lista 1, que também usou 100 observações para treinamento, optando porém pelo algoritmo perceptron. O erro fora da amostra, em ambos os casos, esteve mais próximo da alternativa referente a 1%. O erro médio em 1000 experimentos foi inferior utilizando o algoritmo perceptron.

A média dos erros, de aproximadamente 4,6%, está mais próxima da alternativa c.

Resposta: letra c

Questão 7

Algoritmo perceptron - escolhendo ponto aleatório classificado erradamente a cada iteração. Peso dos coeficientes "default" é um vetor de zero's.

```r
perceptron <- function(X, sinal, taxa_aprendizagem = 1, pesos_perceptron = 0) {

  #adicionar coordanada artificial "x0 = 1"
  X <- cbind(1, X)
  # se não for fornecido peso inicial, começar com zero's
  if (length(pesos_perceptron) == 1 && pesos_perceptron == 0){
    pesos_perceptron <- rep(0, ncol(X))}

  # inicializar número de iterações
  iters <- 0

  #iniciar vetor de sinais preditos com 0 conforme enunciado
  sinal_pred <- X %*% pesos_perceptron
  sinal_pred <- ifelse(sinal_pred > 0, 1, -1)

  # enquanto algum sinal predito for diferente do verdadeiro
  while (any(sinal_pred != sinal)) {

    #encontrar dados classificados erroneamente
    classificado_errado <- which(sinal != sinal_pred)
    # condição utilizada para garantir que não haja iteração extra.
    if (length(classificado_errado) > 0) {
```

```r
      # sortear um dado classificado erradamente
      i <- classificado_errado[sample.int(n = length(classificado_errado),
                                         size = 1)]
      # atualizar vetor de peso
      pesos_perceptron <- pesos_perceptron +
                          taxa_aprendizagem * sinal[i] * X[i,]
      # atualizar número de iterações
      iters <- iters + 1
    }
    #atualizar sinais
    sinal_pred <- X %*% pesos_perceptron
    sinal_pred <- ifelse(sinal_pred > 0, 1, -1)
  }

  # retornar vetor de pesos final e número de iterações
  return(list(pesos_perceptron = pesos_perceptron, iters = iters))
}
```

```r
iters <- c()
set.seed(1)
f_intercepto <- c()
f_coef_angular <- c()
num_experimentos <- 1000
for (i in 1:num_experimentos){
  f <- gerar_f()
  f_intercepto[i] <- f$b
  f_coef_angular[i] <- f$a
  X <- gerar_dados(10)
  sinal <- avaliar_dados(X, avaliar_coef_angular = f_coef_angular[i],
                         avaliar_intercepto = f_intercepto[i])
  pesos_regressao <- lm(sinal ~ X[,1] + X[,2])$coefficients
  iters[i] <- perceptron(X ,sinal, pesos = pesos_regressao)$iters
}
summary(iters)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   0.000   0.000   0.000   4.134   2.000 109.000
```

Aqui vale a pena compararmos com o resultado encontrado na lista 1, iniciando os pesos com um vetor de coordenadas de valor 0. O valor encontrado para a média foi superior a 10, estando mais próximo de 15 que de 1.

Agora, inicializando com os pesos encontrados na regressão linear o algoritmo converge mais rápido. Vemos que o valor encontrado para o terceiro quartil é 2, ou seja, 2 iterações são suficiente para pelo menos 75% dos casos em nosso exemplo.

A média é suscetível a altas variações devido a outliers, razão pela qual neste exemplo encontramos a média superior a 4 mesmo com a mediana sendo 0. É possível notar que houve outliers pois o máximo encontrado foi 109.

A combinação dos algoritmos perceptron e de regressão linear se mostra eficaz para melhorar o tempo de convergência de dados linearmente seperáveis.

O número de iterações está mais próximo da alternativa a, que seria apenas 1 iteração.

Resposta: letra a

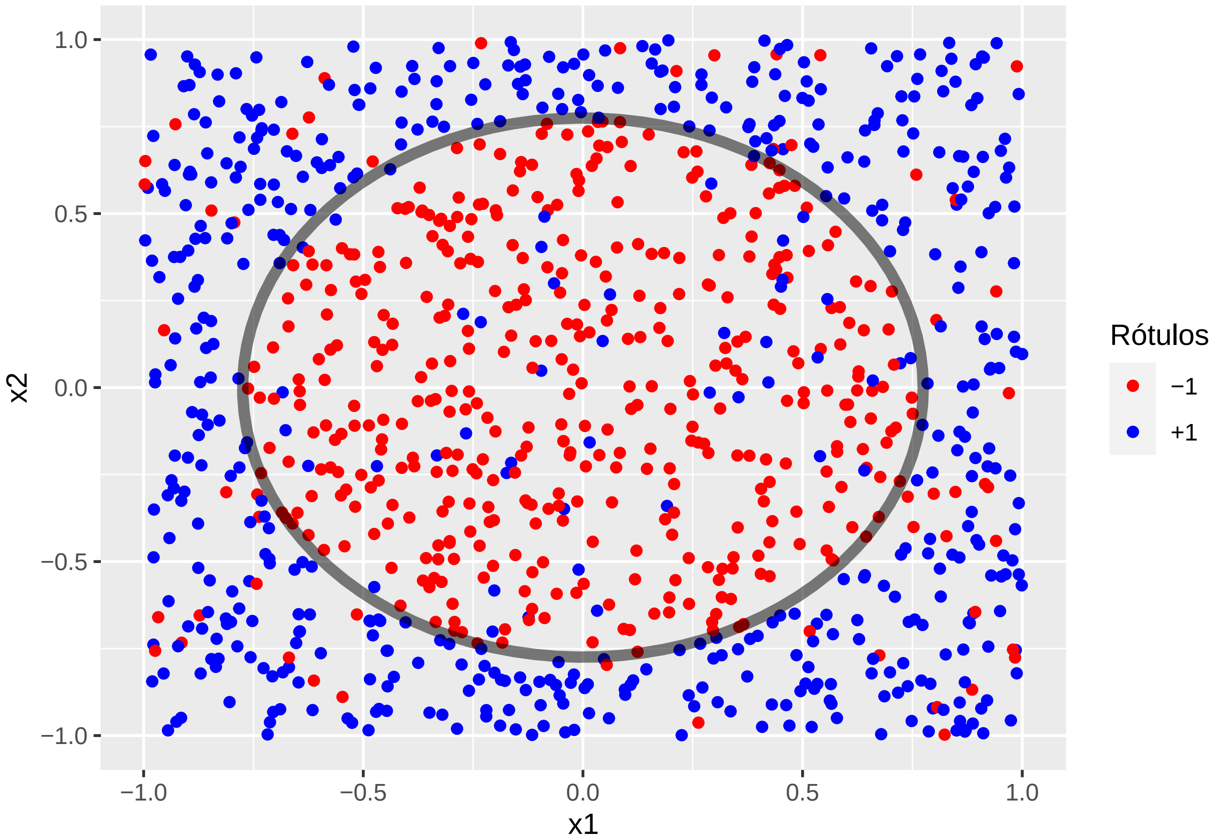
Questão 8

```
funcao_alvo <- function(x1, x2) {
  sign(x1**2 + x2**2 - 0.6)
}
```

```
calcular_erro <- function(X, y, w) {
  # Adicionar coordenada artificial x0 = 1
  X <- cbind(1, X)
  # Prever sinais
  predicoes <- sign(X %*% w)
  # Total de pontos classificados erradamente sobre total de pontos
  erro <- length(which(predicoes != y)) / length(y)
  return(erro)
}
```

Ilustrando o problema

```
set.seed(1)
N <- 1000
X <- gerar_dados(N)
y <- funcao_alvo(X[, 1], X[, 2])
# Selecionar pontos com ruído aleatoriamente
indices_ruido <- sample(1:N, size = floor(N * 0.1))
# Alterar sinal de pontos com ruído
y[indices_ruido] <- -y[indices_ruido]

X <- cbind(1, X)
df <- data.frame(X, y)
df$rotulos <- as.factor(df$y)

ggplot(df, aes(x = x1, y = x2, color = rotulos)) +
      geom_point() +
      scale_color_manual(values = c("red", "blue"), labels = c("-1", "+1")) +
      labs(color = "Rótulos") +
      guides(color = guide_legend(override.aes =
                                  list(fill = c("red","blue")))) +
  geom_path(
    data = data.frame(x1 = cos(seq(0, 2*pi, length.out = 100))*sqrt(0.6),
                      x2 = sin(seq(0, 2*pi, length.out = 100))*sqrt(0.6)),
    aes(x = x1, y = x2),
    col = "black", linewidth = 2, alpha = .5) +
    theme_gray()
```



Primeiro vemos os nossos dados, antes da predição. Os rótulos apresentados são os valores verdadeiros.

Agora ilustraremos o problema com duas tentativas de classificação utilizando regressão linear sem transformação.

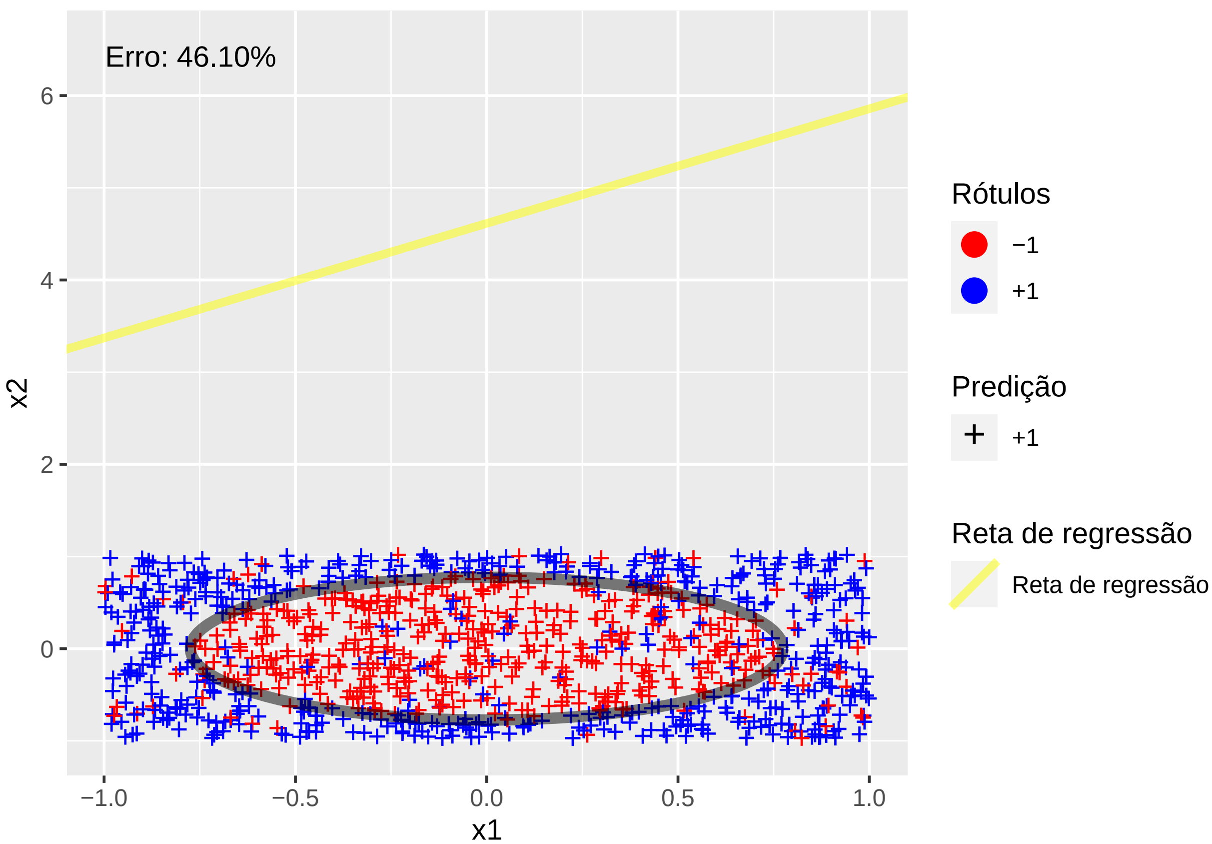
```
plot_exemplo <- function(N){
# Gerar dados
X <- gerar_dados(N)
# Aplicar função alvo nos dados
y <- funcao_alvo(X[, 1], X[, 2])
# Selecionar pontos com ruído
indices_ruido <- sample(1:N, size = floor(N * 0.1))
# Trocar sinal de pontos com ruído
y[indices_ruido] <- -y[indices_ruido]
#Aplicar regressão linear
reg <- lm(y ~ X)
# Armazenar coeficientes da regressão
pesos <- reg$coefficients
print(paste("vetor de pesos: w0: ", pesos[1],
            "w1: ", pesos[2], "w2: ", pesos[3]))
#Calcular erro
erro <- calcular_erro(X, y, pesos)
#Encontrar intercepto e coeficiente angular da retar de regressão
intercepto <- -pesos[1]/pesos[3]
print(paste("intercepto: ", intercepto))
coef_angular <- -pesos[2]/pesos[3]
```

```r
print(paste("coeficiente angular: ", coef_angular))
# Adicionar coordenada artificial "x0 = 1"
X <- cbind(1, X); colnames(X)[1] <- "x0"
# Preparar dataframe para plot
df <- data.frame(X, y)
df$rotulos <- as.factor(df$y)
# Prever classificação com modelo de regressão linear
df$predicao <- as.factor(sign(X %*% pesos))
print("Visualizando as primeiras observações")
print(head(df))
# Ajustes para que o plot funcione corretamente
if (length(unique(df$predicao)) == 1 && unique(df$predicao) == 1) {
  shape_values <- "+"
  labels_values <- "+1"
} else if (length(unique(df$predicao)) == 1 && unique(df$predicao) == -1) {
  shape_values <- "-"
  labels_values <- "-1"
} else {
  shape_values <- c("-", "+")
  labels_values <- c("-1", "+1")
}

# Dataframe auxiliar para plot do círculo
circulo <- data.frame(
  x1 = cos(seq(0, 2 * pi, length.out = 100)) * sqrt(0.6),
  x2 = sin(seq(0, 2 * pi, length.out = 100)) * sqrt(0.6)
)

# Visualização
ggplot(df, aes(x = x1, y = x2, color = rotulos, shape = predicao)) +
  geom_point(size = 4) +
  labs(color = "Rótulos", shape = "Predição", linetype = "Linetype") +
  geom_abline(linewidth = 1.5, alpha = 0.5, color = "yellow", aes(linetype = "Reta de regressão",
              slope = coef_angular,
              intercept = intercepto)) +
  ylim(c(min(-1, intercepto - coef_angular, intercepto + coef_angular),
         max(1, intercepto + coef_angular, intercepto - coef_angular) +
           (max(1, intercepto + coef_angular, intercepto - coef_angular) -
              min(-1, intercepto - coef_angular,
                  intercepto + coef_angular)) / 10)) +
  scale_color_manual(values = c("red", "blue"), labels = c("-1", "+1")) +
  scale_shape_manual(values = shape_values, labels = labels_values) +
  scale_linetype_manual(values = c("solid"),
                        name = "Reta de regressão") +
  geom_path(data = circulo, aes(x = x1, y = x2), col = "black",
            linewidth = 2, alpha = 0.5, inherit.aes = FALSE) +
  annotate("text", x = min(df$x1),
      y = max(1, intercepto + coef_angular, intercepto - coef_angular) +
          (max(1, intercepto + coef_angular, intercepto - coef_angular) -
          min(-1, intercepto - coef_angular, intercepto + coef_angular)) / 10,
           label = paste("Erro:", sprintf("%.2f%%", erro * 100)),
           hjust = 0, vjust = 1, color = "black") +
  guides(color = guide_legend(order = 1),
```

```r
        shape = guide_legend(order = 2,
                   override.aes = list(linetype = NA, size = 5)),
        linetype = guide_legend(order = 3,
                   override.aes = list(color = c("yellow"),shape = c(NA)))) +
  theme_gray()
}
```

```r
set.seed(1)
plot_exemplo(1000)
```

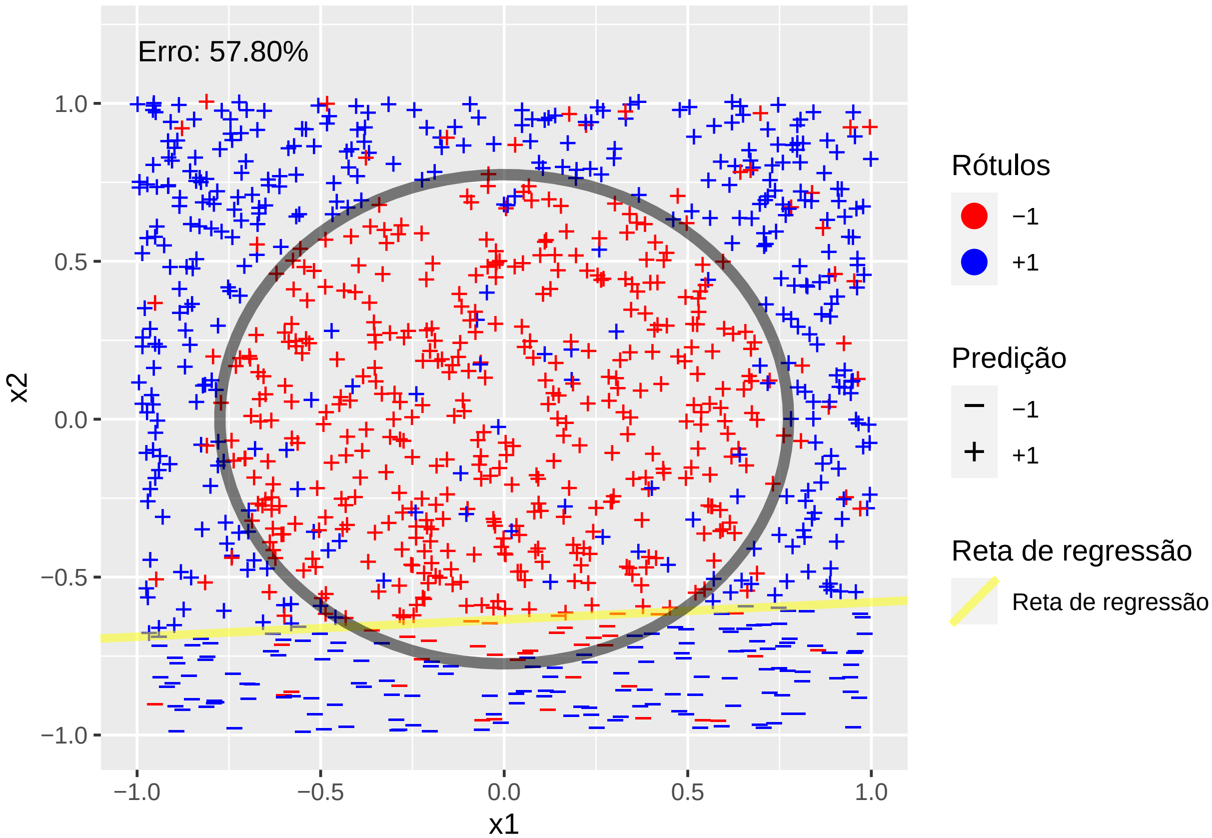
```
## [1] "vetor de pesos: w0:  0.0776861295564353 w1:  0.0209306470045088 w2:  -0.0168375479040109"
## [1] "intercepto:  4.61386242220755"
## [1] "coeficiente angular:  1.2430935385504"
## [1] "Visualizando as primeiras observações"
##    x0         x1          x2  y rotulos predicao
## p1  1 -0.4689827  0.06161759 -1      -1        1
## p2  1 -0.2557522  0.36972181 -1      -1        1
## p3  1  0.1457067 -0.23343321 -1      -1        1
## p4  1  0.8164156  0.90997600  1       1        1
## p5  1 -0.5966361 -0.76328684  1       1        1
## p6  1  0.7967794 -0.92179989  1       1        1
```



Neste exemplo o modelo classificou todos os sinais como positivos. O erro encontrado foi de 46.10%.

```r
set.seed(2)
plot_exemplo(1000)
```

```
## [1] "vetor de pesos: w0:  0.0674153204403092 w1:  -0.00580785768244662 w2:  0.106294551093462"
## [1] "intercepto:  -0.634231197618331"
## [1] "coeficiente angular:  0.054639279461653"
## [1] "Visualizando as primeiras observações"
##    x0         x1          x2  y rotulos predicao
## p1  1 -0.6302355 -0.35433261 -1      -1        1
## p2  1  0.4047481 -0.91063350  1       1       -1
## p3  1  0.1466527  0.46337595 -1      -1        1
## p4  1 -0.6638962 -0.01515374 -1      -1        1
## p5  1  0.8876787 -0.52873501  1       1        1
## p6  1  0.8869499 -0.74793099  1       1       -1
```



No segundo exemplo, embora o modelo tenha classificado parte dos sinais como positivos e outra parte como negativos, o resultado foi pior, com um erro de aproximadamente 57%. O intercepto de -0.634231197618331 e o coeficiente angular de 0.0546392794616529, bem próximo a 0, fazem com que a reta de regressão encontrada classifique como negativo majoritariamente pontos abaixo do círculo.

No entanto, pontos fora do círculo, salvo ruído, são positivos. Isso faz com que este modelo erre mais que metade das vezes, que seria o esperado para uma reta dividindo [-1,1] x [-1,1] em partes de áreas iguais (como a reta x1 = 0, reta x2 = 0 ou x2 = x1).

Agora iremos repetir o experimento 1000 vezes para encontrar a resposta da questão.

```r
# Fixar seed para garantir reprodutibilidade
set.seed(1)
# tamanho da amostra
N <- 1000
```

```r
# número de repetições do experimento
num_experimentos <- 1000
# inicializar erro total dos experimentos
soma_erro <- 0
# Para cada experimento
for (i in 1:num_experimentos) {
  # Gerar dados
  X <- gerar_dados(N)
  # Aplicar função alvo aos dados
  y <- funcao_alvo(X[, 1], X[, 2])
  # Selecionar pontos com ruido aleatoriamente
  indices_ruido <- sample(1:N, size = floor(N * 0.1))
  # Alterar sinal de pontos com ruído
  y[indices_ruido] <- -y[indices_ruido]
  # Aplicar regressão linear
  reg <- lm(y ~ X)
  # Armazenar coeficientes da regressão
  w <- reg$coefficients
  # Calcular erro
  erro <- calcular_erro(X, y, w)
  # Atualizar erro acumulado nos experimentos
  soma_erro <- soma_erro + erro
}
# Calcular erro médio na amostra
erro_medio <- soma_erro / num_experimentos
```

```r
print(paste("Erro médio na amostra:", erro_medio))
```

```
## [1] "Erro médio na amostra: 0.503694999999999"
```

O erro médio é próximo de 50%, a alternativa correta é a letra d.

Resposta: letra d

Questão 9

```r
# Fixar seed para reprodutibilidade
set.seed(1)
# Número de observações na amostra
N <- 1000
# Gerar dados
X <- gerar_dados(N)
# Adicionar coordenadas
vetor_adicional_1 <- X[,1] * X[,2]
vetor_adicional_2 <- X[,1] ** 2
vetor_adicional_3 <- X[,2] ** 2
X <- cbind(X, vetor_adicional_1, vetor_adicional_2, vetor_adicional_3)
colnames(X) <- c("x1", "x2","x1*x2","x1²", "x2²")
print("Visualizando primeiras observações")
```

```
## [1] "Visualizando primeiras observações"
```

```r
print(head(X))
```

```
##              x1          x2       x1*x2         x1²         x2²
```

```
## p1 -0.4689827  0.06161759 -0.02889758 0.21994475 0.003796727
## p2 -0.2557522  0.36972181 -0.09455717 0.06540919 0.136694215
## p3  0.1457067 -0.23343321 -0.03401279 0.02123045 0.054491065
## p4  0.8164156  0.90997600  0.74291858 0.66653440 0.828056315
## p5 -0.5966361 -0.76328684  0.45540451 0.35597468 0.582606798
## p6  0.7967794 -0.92179989 -0.73447114 0.63485736 0.849715036
```

```
# Aplicar função alvo nos dados
y <- funcao_alvo(X[, 1], X[, 2])
# Selecionar pontos com ruído
indices_ruido <- sample(1:N, size = floor(N * 0.1))
# Trocar sinal de pontos com ruído
y[indices_ruido] <- -y[indices_ruido]
```

```
reg <- lm(y~X)
X <- cbind(1, X); colnames(X)[1] <- "x0"

print("Visualizando as primeiras observações")
```

```
## [1] "Visualizando as primeiras observações"
```

```
print(head(cbind(X,y)))
```

```
##    x0         x1          x2       x1*x2        x1²          x2²  y
## p1  1 -0.4689827  0.06161759 -0.02889758 0.21994475 0.003796727 -1
## p2  1 -0.2557522  0.36972181 -0.09455717 0.06540919 0.136694215 -1
## p3  1  0.1457067 -0.23343321 -0.03401279 0.02123045 0.054491065 -1
## p4  1  0.8164156  0.90997600  0.74291858 0.66653440 0.828056315  1
## p5  1 -0.5966361 -0.76328684  0.45540451 0.35597468 0.582606798  1
## p6  1  0.7967794 -0.92179989 -0.73447114 0.63485736 0.849715036  1
```

```
pesos <- reg$coefficients; names(pesos) <- c("w0","w1","w2","w3","w4","w5")
pesos
```

```
##          w0          w1          w2          w3          w4          w5
## -0.95844795 -0.04988209  0.02391024  0.04672071  1.38129078  1.64620132
```

Apenas pelos coeficientes encontrados já é possível observar um comportamento esperado. Os coeficientes relativos a x0, $x1^2$ e $x2^2$ são os maiores, enquanto os demais estão próximos de 0, compatível com nossa função alvo ($x1^2 + x2^2 = 0.6$).

```
predicao <- sign(X %*% pesos)
erro <- length(which(predicao != y))
print(paste("Erro encontrado no exemplo: ", sprintf("%.2f%%", erro/N * 100)))
```

```
## [1] "Erro encontrado no exemplo:  12.00%"
```

Neste caso particular, foi encontrado um erro de 12%. Dado que temos um ruído em 10% dos nossos dados, o valor encontrado está de acordo com o esperado dadas as observações dos pesos encontrados.

Agora iremos responder a pergunta da questão, repetindo o processo 1000 vezes e comparando com as opções.

```
pesos_a <- c(-1, -0.05, 0.08, 0.13, 1.5, 1.5)
pesos_b <- c(-1, -0.05, 0.08, 0.13, 1.5, 15)
pesos_c <- c(-1, -0.05, 0.08, 0.13, 15, 15)
pesos_d <- c(-1, -1.5, 0.08, 0.13, 0.05, 0.05)
pesos_e <- c(-1, -0.05, 0.08, 1.5, 0.15, 0.15)
```

```r
# Fixar seed para garantir reprodutibilidade
set.seed(1)
# tamanho da amostra
N <- 1000
set.seed(1)
N <- 1000
# número de experimentos
num_experimentos <- 1000

#inicializar erro
soma_erro <- 0
#inicializar diferencas com as opcoes do enunciado
diferenca_a <- 0
diferenca_b <- 0
diferenca_c <- 0
diferenca_d <- 0
diferenca_e <- 0

# Para cada experimento
for (i in 1:num_experimentos) {
  # Gerar dados
  X <- gerar_dados(N)
  # Adicionar coordenadas
  vetor_adicional_1 <- X[,1] * X[,2]
  vetor_adicional_2 <- X[,1] ** 2
  vetor_adicional_3 <- X[,2] ** 2
  X <- cbind(X, vetor_adicional_1, vetor_adicional_2, vetor_adicional_3)
  colnames(X) <- c("x1", "x2","x1*x2","x1²", "x2²")
  # Aplicar função alvo aos dados
  y <- funcao_alvo(X[, 1], X[, 2])
  # Selecionar pontos com ruido aleatoriamente
  indices_ruido <- sample(1:N, size = floor(N * 0.1))
  # Alterar sinal de pontos com ruído
  y[indices_ruido] <- -y[indices_ruido]
  # Aplicar regressão linear
  reg <- lm(y ~ X)
  # Armazenar coeficientes da regressão
  pesos <- reg$coefficients
  # Adicionar coordenada artificial "x0 = 1 "
  X <- cbind(1,X)
  # Prever sinais
  predicao <- sign(X %*% pesos)
  # Calcular erro
  erro <- length(which(predicao != y))
  #Calcular diferenças com as opções
  dif_experimento_a <- length(which(predicao != sign(X%*%pesos_a)))
  dif_experimento_b <- length(which(predicao != sign(X%*%pesos_b)))
  dif_experimento_c <- length(which(predicao != sign(X%*%pesos_c)))
  dif_experimento_d <- length(which(predicao != sign(X%*%pesos_d)))
  dif_experimento_e <- length(which(predicao != sign(X%*%pesos_e)))
  # Atualizar erro e diferenças totais nos experimentos
  soma_erro <- soma_erro + erro
  diferenca_a <- diferenca_a + dif_experimento_a
```

```
  diferenca_b <- diferenca_b + dif_experimento_b
  diferenca_c <- diferenca_c + dif_experimento_c
  diferenca_d <- diferenca_d + dif_experimento_d
  diferenca_e <- diferenca_e + dif_experimento_e
}
print(paste("Erro médio na amostra:",
            sprintf("%.2f%%", soma_erro/N/num_experimentos * 100)))
```

```
## [1] "Erro médio na amostra: 12.39%"
```

```
print(paste("Diferença média com opção a:",
            sprintf("%.2f%%", diferenca_a/N/num_experimentos * 100)))
```

```
## [1] "Diferença média com opção a: 3.86%"
```

```
print(paste("Diferença média com opção b:",
            sprintf("%.2f%%", diferenca_b/N/num_experimentos * 100)))
```

```
## [1] "Diferença média com opção b: 33.70%"
```

```
print(paste("Diferença média com opção c:",
            sprintf("%.2f%%", diferenca_c/N/num_experimentos * 100)))
```

```
## [1] "Diferença média com opção c: 44.78%"
```

```
print(paste("Diferença média com opção d:",
            sprintf("%.2f%%", diferenca_d/N/num_experimentos * 100)))
```

```
## [1] "Diferença média com opção d: 36.82%"
```

```
print(paste("Diferença média com opção e:",
            sprintf("%.2f%%", diferenca_e/N/num_experimentos * 100)))
```

```
## [1] "Diferença média com opção e: 43.92%"
```

Dentre as opções da questão, a mais próxima ao nosso modelo previsto é a alternativa a.

Resposta: letra a

Questão 10

```
# Fixar seed para garantir reprodutibilidade
set.seed(1)
# tamanho da amostra
N <- 1000
# número de experimentos
num_experimentos <- 1000

#inicializar erros
soma_erro_dentro <- 0
soma_erro_fora <- 0

# Para cada experimento
for (i in 1:num_experimentos) {
  # Gerar dados
  X <- gerar_dados(N*2)
  # Adicionar coordenadas
```

```r
  vetor_adicional_1 <- X[,1] * X[,2]
  vetor_adicional_2 <- X[,1] ** 2
  vetor_adicional_3 <- X[,2] ** 2
  X <- cbind(X, vetor_adicional_1, vetor_adicional_2, vetor_adicional_3)
  colnames(X) <- c("x1", "x2","x1*x2","x1²", "x2²")
  # Aplicar função alvo aos dados
  y <- funcao_alvo(X[, 1], X[, 2])

  ##########Treino##########
  X_treino <- X[1:N,]
  y_treino <- y[1:N]
  # Selecionar pontos com ruido aleatoriamente
  indices_ruido_treino <- sample(1:N, size = floor(N * 0.1))
  # Alterar sinal de pontos com ruído
  y_treino[indices_ruido] <- -y_treino[indices_ruido]
  # Aplicar regressão linear
  reg <- lm(y_treino ~ X_treino)
  # Armazenar coeficientes da regressão
  pesos_treino <- reg$coefficients
  # Adicionar coordenada artificial "x0 = 1 "
  X_treino <- cbind(1,X_treino)
  # Prever sinais
  predicao <- sign(X_treino %*% pesos_treino)
  # Calcular erro
  erro_dentro <- length(which(predicao != y_treino))
  # Atualizar erro dentro
  soma_erro_dentro <- soma_erro_dentro + erro_dentro

  ##########Teste##########
  # Fora da amostra
  X_teste <- X[(N + 1):(2*N),]
  y_teste <- y[(N + 1):(2*N)]
  # Selecionar pontos com ruido aleatoriamente
  indices_ruido_teste <- sample(1:N, size = floor(N * 0.1))
  # Alterar sinal de pontos com ruído
  y_teste[indices_ruido] <- -y_teste[indices_ruido]
  # adicionar coordenada artificial "x0 = 1"
  X_teste <- cbind(1,X_teste)
  # Adicionar coordenadas
  predicao_teste <- sign(X_teste %*% pesos_treino)
  # Calcular erro
  erro_fora <- length(which(predicao_teste != y_teste))
  # Atualizar fora
  soma_erro_fora <- soma_erro_fora + erro_fora
}
print(paste("Erro médio dentro da amostra:",
  sprintf("%.2f%%", soma_erro_dentro/N/num_experimentos * 100)))
```

```
## [1] "Erro médio dentro da amostra: 12.38%"
```

```r
print(paste("Erro médio fora da amostra:",
  sprintf("%.2f%%", soma_erro_fora/N/num_experimentos * 100)))
```

```
## [1] "Erro médio fora da amostra: 12.62%"
```

O erro médio fora da amostra encontrado foi de 12,62% em 1000 experimentos com 1000 observações em cada um deles. Logo, a opção mais próxima é a letra b, equivalente a 10%. Dado que temos um erro de 10% adicionado pelo ruído, uma resposta próxima a 10% é o desejado.

O erro fora da amostra encontrado foi maior que o erro dentro da amostra, característica esperada em aprendizado de máquina, apesar de não obrigatória. O erro fora da amostra acompanha bem o erro dentro da amostra, uma vez que estão razoavelmente próximos.

Resposta: letra b

Questão 11

Crie um conjunto de dados linearmente separável com 100 pontos em R^2, utilizando o mesmo procedimento descrito nas questões práticas do Perceptron da Lista 1.

```r
set.seed(123)
f <- gerar_f()
print(f)
```

```
## $x1
## [1] -0.4248450  0.5766103
##
## $x2
## [1] -0.1820462  0.7660348
##
## $a
## [1] 0.7801708
##
## $b
## [1] 0.9080619
```

```r
set.seed(123)
N <- 100
X <- gerar_dados(N)
sinal <- avaliar_dados(X, avaliar_intercepto = f$b, avaliar_coef_angular = f$a)
print("Visualizando as primeiras observações")
```

```
## [1] "Visualizando as primeiras observações"
```

```r
head(X)
```

```
##            x1          x2
## p1 -0.4248450  0.19997792
## p2  0.5766103 -0.33435292
## p3 -0.1820462 -0.02277393
## p4  0.7660348  0.90894765
## p5  0.8809346 -0.03419521
## p6 -0.9088870  0.78070044
```

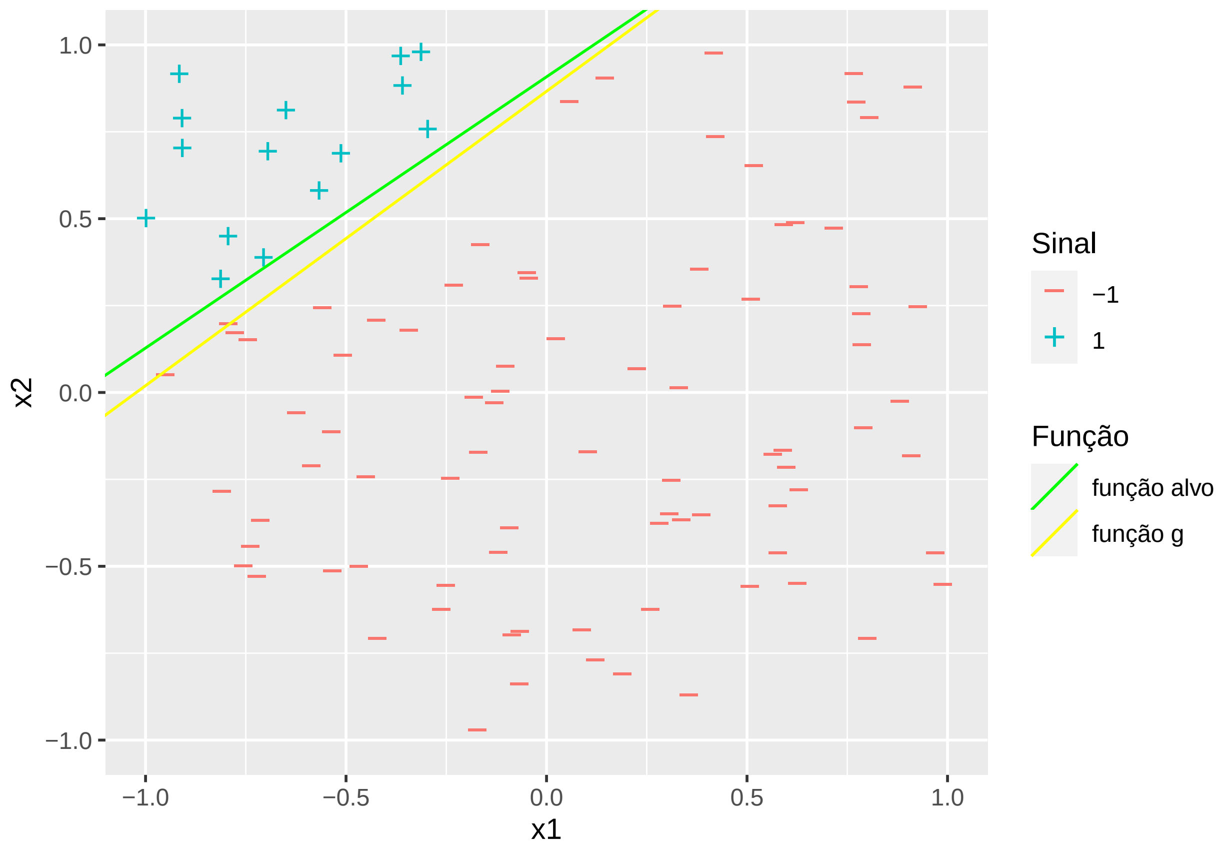
```r
set.seed(123)
w <- perceptron(X,sinal)$pesos_perceptron
coef_angular <- -w[2] / w[3]
intercepto <- -w[1]/ w[3]
print(paste("Coeficiente angular: ", coef_angular))
```

```
## [1] "Coeficiente angular:  0.846750316345266"
```

```r
print(paste("Intercepto: ", intercepto))
```

```
## [1] "Intercepto:  0.866391973779336"
```

```r
# Visualização do problema
library(ggplot2)

# Criar dataframe com dados e sinais
df <- data.frame(X, sinal)

# Plotar pontos, função alvo e reta dada pelos coeficientes do perceptron
ggplot(df, aes(x=X[,1], y=X[,2], size = 3,color=factor(sinal),
               shape = factor(sinal))) +
  geom_point(show.legend = TRUE) +
  xlim(-1, 1) + ylim(-1, 1) +
  geom_abline(aes(slope = f$a, intercept = f$b, linetype = "função alvo"),
              color = "green") +
  geom_abline(aes(slope = coef_angular, intercept = intercepto,
                  linetype = "função g"), color = "yellow") +
  scale_shape_manual(values = c("-", "+"), name = "Sinal") +
  scale_linetype_manual(values = c("solid", "solid"), name = "Função") +
  guides(color = guide_legend(override.aes =
                    list(shape = c("-", "+"), size = 5))) +
  guides(linetype = guide_legend(override.aes =
                    list(color = c("green", "yellow"),shape = c(NA, NA)))) +
  labs(color = "Sinal", shape = "Sinal",
       linetype = "Função", x = "x1", y = "x2") +
  scale_size(guide = FALSE)
```



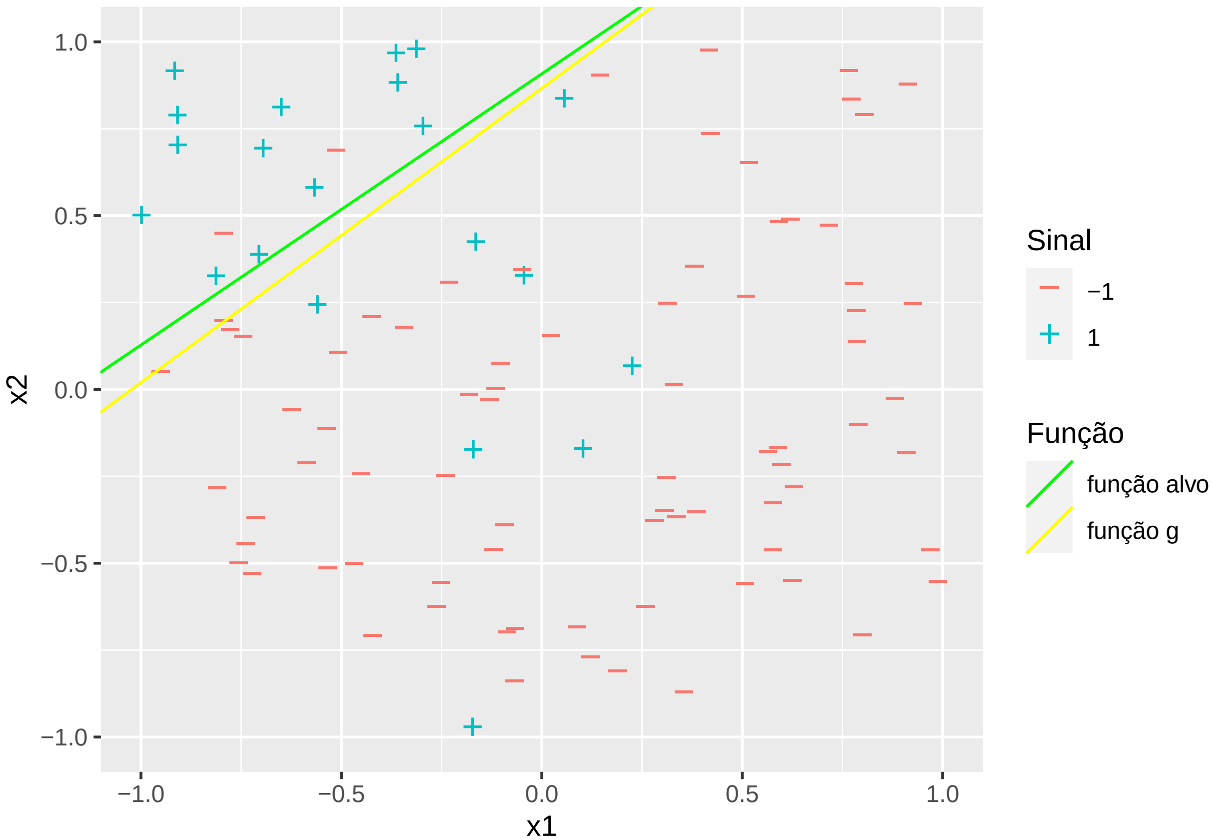
Até aqui o problema é linearmente separável, e encontramos uma função g que satisfaz esta condição.

Agora, selecione aleatoriamente 10% dos pontos e inverta os rótulos dos pontos selecionados, efetivamente transformando o conjunto de dados em não-linearmente separável.

```
# Selecionar pontos com ruido aleatoriamente
indices_ruido <- sample(1:N, size = floor(N * 0.1))
# Alterar sinal de pontos com ruído
sinal[indices_ruido] <- - sinal[indices_ruido]
```

```
# Visualização do problema
library(ggplot2)

# Criar dataframe com dados e sinais
df <- data.frame(X, sinal)

# Plotar pontos, função alvo e reta dada pelos coeficientes do perceptron
ggplot(df, aes(x=X[,1], y=X[,2], size = 3,
               color=factor(sinal), shape = factor(sinal))) +
  geom_point(show.legend = TRUE) +
  xlim(-1, 1) + ylim(-1, 1) +
  geom_abline(aes(slope = f$a, intercept = f$b, linetype = "função alvo"),
              color = "green") +
  geom_abline(aes(slope = coef_angular, intercept = intercepto,
```

```r
                  linetype = "função g"), color = "yellow") +
  scale_shape_manual(values = c("-", "+"), name = "Sinal") +
  scale_linetype_manual(values = c("solid", "solid"), name = "Função") +
  guides(color = guide_legend(override.aes =
                      list(shape = c("-", "+"), size = 5))) +
  guides(linetype = guide_legend(override.aes =
                      list(color = c("green", "yellow"),shape = c(NA, NA)))) +
  labs(color = "Sinal", shape = "Sinal",
       linetype = "Função", x = "x1", y = "x2") +
  scale_size(guide = FALSE)
```



Agora a função g encontrada anteriormente não mais é capaz de separar linearmente os dados. Não há garantia que os dados sejam linearmente separáveis ao realizar este experimento.

Em seguida, implemente o algoritmo PLA pocket e treine-o neste conjunto de dados por k iterações. Ao término deste treinamento, gere 1000 pontos e rotule-os de acordo com a função alvo original; use estes pontos para estimar o E_out do pocket.

Algoritmo perceptron pocket

```r
perceptron_pocket <- function(X, sinal, taxa_aprendizagem = 1, k,
                              pesos_perceptron = 0) {
  # Adicionar coordenada artificial "x0 = 1"
  X <- cbind(1, X)
  # se não for fornecido peso inicial, começar com zero's
```

```r
  if (length(pesos_perceptron) == 1 && pesos_perceptron == 0){
    pesos_perceptron <- rep(0, ncol(X))}
  # inicializar número de iterações
  iters <- 0

  #iniciar vetor de sinais preditos com 0 conforme enunciado
  sinal_pred <- X %*% pesos_perceptron
  sinal_pred <- ifelse(sinal_pred > 0, 1, -1)

  # inicializar vetor de melhores pesos e variável menor_erro
  melhores_pesos <- pesos_perceptron
  menor_erro <- length(which((sinal_pred != sinal)))

   # enquanto algum sinal predito for diferente do verdadeiro
  while (iters < k && any(sinal_pred != sinal)) {
    #encontrar dados classificados erroneamente
    classificado_errado <- which(sinal != sinal_pred)
    # condição utilizada para garantir que não haja iteração extra.
    if (length(classificado_errado) > 0) {
      # sortear um dado classificado erradamente
      i <- classificado_errado[sample.int(n = length(classificado_errado),
                               size = 1)]
      # Calcular pesos encontrados na iteração
      pesos_perceptron <- pesos_perceptron +
                          taxa_aprendizagem * sinal[i] * X[i,]
      erro <- length(which((sinal_pred != sinal)))

      # atualizar vetor de peso do problema se o erro diminuir
      if (erro < menor_erro) {
      melhores_pesos <- pesos_perceptron
      menor_erro <- erro
    }
      # atualizar número de iterações
      iters <- iters + 1
    }
    #atualizar sinais
    sinal_pred <- X %*% pesos_perceptron
    sinal_pred <- ifelse(sinal_pred > 0, 1, -1)
  }

  # retornar vetor de pesos, número de iterações, e erro
  return(list(pesos_perceptron = melhores_pesos,
              iters = iters, erro = menor_erro))
}
```

```r
teste_perceptron_pocket_ruido <- function(num_experimentos, n_iter_perceptron,
                                          N_dentro, N_fora , usa_pesos_regressao) {

  # Fixar seed para reprodutibilidade
  set.seed(123)

  # Inicializar vetores de erro
  erro_fora <- c()
  erro_dentro <- c()
```

```r
# Para cada experimento
for (i in 1:num_experimentos) {
  # Gerar função f
  f <- gerar_f()

  # Gerar dados - treino e teste
  X <- gerar_dados(N_dentro + N_fora)

  # Separar dados de treino e teste
  X_dentro <- X[1:N_dentro,]
  X_fora <- X[(N_dentro + 1) : nrow(X),]

  # Avaliar sinal (verdadeiro) dos dados de treinamento
  sinal_dentro <- avaliar_dados(X_dentro, avaliar_intercepto = f$b,
                                avaliar_coef_angular = f$a)

  # Selecionar pontos com ruido aleatoriamente nos dados de treinamento
  indices_ruido_dentro <- sample(1:N_dentro, size = floor(N_dentro * 0.1))

  # Alterar sinal de pontos com ruído
  sinal_dentro[indices_ruido_dentro] <- -sinal_dentro[indices_ruido_dentro]

  # Avaliar sinal (verdadeiro) dos dados de teste
  sinal_fora <- avaliar_dados(X_fora, avaliar_intercepto = f$b,
                              avaliar_coef_angular = f$a)

  if (usa_pesos_regressao == TRUE) {
    # Fazendo regressão linear nos dados de treinamento
    reg <- lm(sinal_dentro ~ X_dentro)

    # Armazenar pesos da regressão
    pesos_regressao <- reg$coefficients

    # Rodar perceptron nos dados de treino com pesos da regressão
    saida_perceptron_pocket <- perceptron_pocket(X_dentro, sinal_dentro,
              k = n_iter_perceptron, pesos_perceptron = pesos_regressao)
  } else {
    # Rodar perceptron nos dados de treino sem pesos da regressão
    saida_perceptron_pocket <- perceptron_pocket(X_dentro, sinal_dentro,
                              k = n_iter_perceptron)
  }

  # Armazenar erro na amostra no experimento
  erro_dentro[i] <- saida_perceptron_pocket$erro

  # Armazenas pesos do perceptron treinado do experimento
  pesos_perceptron_dentro <- saida_perceptron_pocket$pesos_perceptron

  # Prever sinais de teste
  sinal_pred_fora <- sign(cbind(1, X_fora) %*% pesos_perceptron_dentro)

  # Calcular total de observações classificadas erradamento no experimento
  erro_fora[i] <- length(which(sinal_pred_fora != sinal_fora))
```

```r
  }

  # Mostrar resultados
  print(paste("Erro dentro da amostra: ",
              sprintf("%.2f%%", 100 * mean(erro_dentro / N_dentro))))

  print(paste("Erro fora da amostra: ",
              sprintf("%.2f%%", 100 * mean(erro_fora) / N_fora)))

  # retornar vetor de pesos, número de iterações, e erro
  return(list(erro_dentro = summary(erro_dentro),
              erro_fora = summary(erro_fora)))
}
```

11.1) Inicializando os pesos com 0; k = 10;

```r
set.seed(123)
teste_perceptron_pocket_ruido(num_experimentos = 1000, n_iter_perceptron = 10,
                              N_dentro = 100, N_fora = 1000,
                              usa_pesos_regressao = FALSE)
```

```
## [1] "Erro dentro da amostra:  18.05%"
## [1] "Erro fora da amostra:  36.64%"
```

```
## $erro_dentro
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9.00   14.00   17.00   18.05   21.00   42.00
##
## $erro_fora
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    52.0   237.8   314.0   366.4   439.5  1000.0
```

11.2) Inicializando os pesos com 0; k = 50;

```r
set.seed(123)
teste_perceptron_pocket_ruido(num_experimentos = 1000, n_iter_perceptron = 50,
                              N_dentro = 100, N_fora = 1000,
                              usa_pesos_regressao = FALSE)
```

```
## [1] "Erro dentro da amostra:  12.79%"
## [1] "Erro fora da amostra:  28.74%"
```

```
## $erro_dentro
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    8.00   11.00   12.00   12.79   14.00   24.00
##
## $erro_fora
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    21.0   192.0   268.0   287.4   356.0  1000.0
```

11.3) Inicializando os pesos usando Regressão Linear; k = 10;

```r
set.seed(123)
teste_perceptron_pocket_ruido(num_experimentos = 1000, n_iter_perceptron = 10,
```

```r
                              N_dentro = 100, N_fora = 1000,
                              usa_pesos_regressao = TRUE)
```

```
## [1] "Erro dentro da amostra:  13.09%"
## [1] "Erro fora da amostra:  13.57%"

## $erro_dentro
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    8.00   11.00   13.00   13.09   15.00   22.00
##
## $erro_fora
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0    35.0    61.0   135.7   120.0   942.0
```

11.4) Inicializando os pesos usando Regressão Linear; k = 50.

```r
set.seed(123)
teste_perceptron_pocket_ruido(num_experimentos = 1000, n_iter_perceptron = 50,
                              N_dentro = 100, N_fora = 1000,
                              usa_pesos_regressao = TRUE)
```

```
## [1] "Erro dentro da amostra:  11.89%"
## [1] "Erro fora da amostra:  22.29%"

## $erro_dentro
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    7.00   10.00   12.00   11.89   13.00   21.00
##
## $erro_fora
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0    46.0   159.5   222.9   341.5   972.0
```

Primeiramente, comparando os erros dentro da amostra, vemos que eles são reduzidos tanto quanto utilizamos o vetor de pesos iniciais da regressão linear tanto quanto aumentamos o número de iterações do perceptron pocket.

O erro fora da amostra é reduzido quando optamos por utilizar os pesos da regressão. No entanto, nos casos que são utilizados pesos da regressão no perceptron pocket, o erro fora da amostra aumentou quando o número de iterações do perceptron aumentou. Isso ocorreu embora o erro dentro da amostra tenha diminuído. Isso mostra que o erro fora da amostra não acompanhou o erro dentro da amostra, indicando que o modelo ficou sobreajustado aos dados de treinamento, não sendo capaz de fazer uma boa generalização.

A utilização dos pesos da regressão foi capaz de reduzir ambos os erros quando nos casos com 10 iterações.

Dentre as opções apresentadas, o melhor modelo aparenta ser o terceiro, com k = 10 e usando os pesos da regressão, pois possui o menor erro fora da amostra e o erro fora da amostra parece estar acompanhando o erro dentro da amostra.

Vale ressaltar que esta "escolha" se mostra ideal ainda que os erros dentro da amostra dos modelos 2 e 4 sejam menores, pois o objetivo do aprendizado de máquina é generalizar para novos dados, diferente dos que foram usados no treinamento.

Uma observação é que, como não há ruído nos dados de teste, eles são linearmente separáveis. De fato, no quarto caso o erro mínimo encontrado foi 0, havendo portanto pelo menos uma ocasião que o modelo separou os dados sem nenhum erro. Isso não significa que o modelo irá separá-los precisamente, pois para que isso ocorra precisa coincidir dos coeficientes gerados no perceptron pocket treinados com os dados ruidosos coincidirem com uma das possibilidades de coeficientes que separaria os dados de teste.