

## Lista 5

César A. Galvão - 190011572

Gabriela Carneiro - 180120816

João Vitor Vasconcelos - 170126064

Kevyn Andrade de Souza - 190015853

## Índice

<b>Questão 12</b>	<b>2</b>
Questão 1 . . . . .	2
Questão 2 . . . . .	4
Questão 3 . . . . .	7
Questão 4 . . . . .	13
Questão 6 . . . . .	18
Questão 7 . . . . .	26
<b>Questão 13</b>	<b>30</b>

## Questão 12

Revise as notas de aula e estude o Capítulo 9 de James et al. (with Applications in R ou with Applications in Python), disponível em <https://www.statlearning.com/>. Resolva os exercícios deste capítulo.

---

### Questão 1

This problem involves hyperplanes in two dimensions.

#### Item a)

Sketch the hyperplane  $1 + 3X_1 - X_2 = 0$ . Indicate the set of points for which  $1 + 3X_1 - X_2 > 0$ , as well as the set of points for which  $1 + 3X_1 - X_2 < 0$ .

---

A Figura 1 mostra o hiperplano indicado, assim como as curvas de nível. Os pontos em que o valor da função que gera o hiperplano é maior que zero estão à direita da curva de nível com valor 0 e os demais estão à esquerda.

```
data <- data.frame(
  x1 = seq(-5, 5, length.out = 100),
  x2 = seq(-5, 5, length.out = 100)
)

ggplot(data, aes(x = x1, y = x2)) +
  geom_abline(intercept = 1, slope = 3, color = "blue") +
  geom_ribbon(aes(ymin = -Inf, ymax = (3*x1 + 1)), xmin = min(data$x1), xmax = Inf,
  ↪ fill = "blue", alpha = 0.2) +
  scale_y_continuous(expand = c(0,0)) +
  scale_x_continuous(expand = c(0,0)) +
  labs(x = TeX("\$x_1\$"), y = TeX("\$x_2\$"))+
  annotate("text", x = 2, y = -5, label = "y > 0", color = "red", size = 5) +
  theme_classic()+
  theme(panel.grid.major = element_line(),
    axis.title.y = element_text(angle = 0, vjust = 0.5))
```

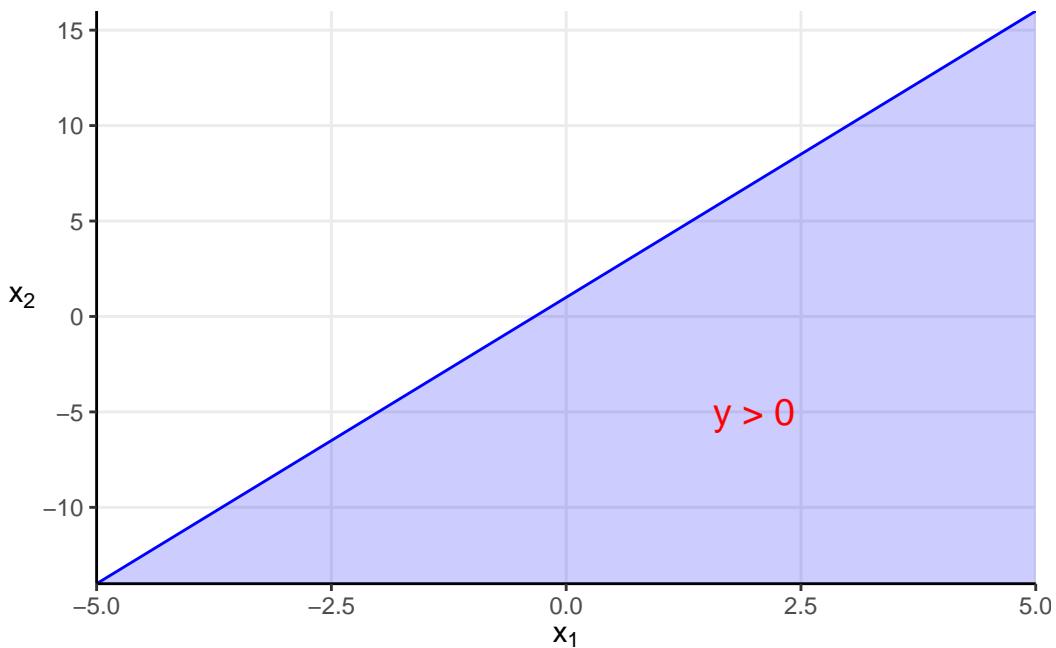


Figura 1: Hiperplano  $1 + 3X_1 - X_2 = 0$

**Item b)**

On the same plot, sketch the hyperplane  $-2 + X_1 + 2X_2 = 0$ . Indicate the set of points for which  $-2 + X_1 + 2X_2 > 0$ , as well as the set of points for which  $-2 + X_1 + 2X_2 < 0$ .

---

A Figura 2 mostra o hiperplano indicado da mesma forma que no item anterior.

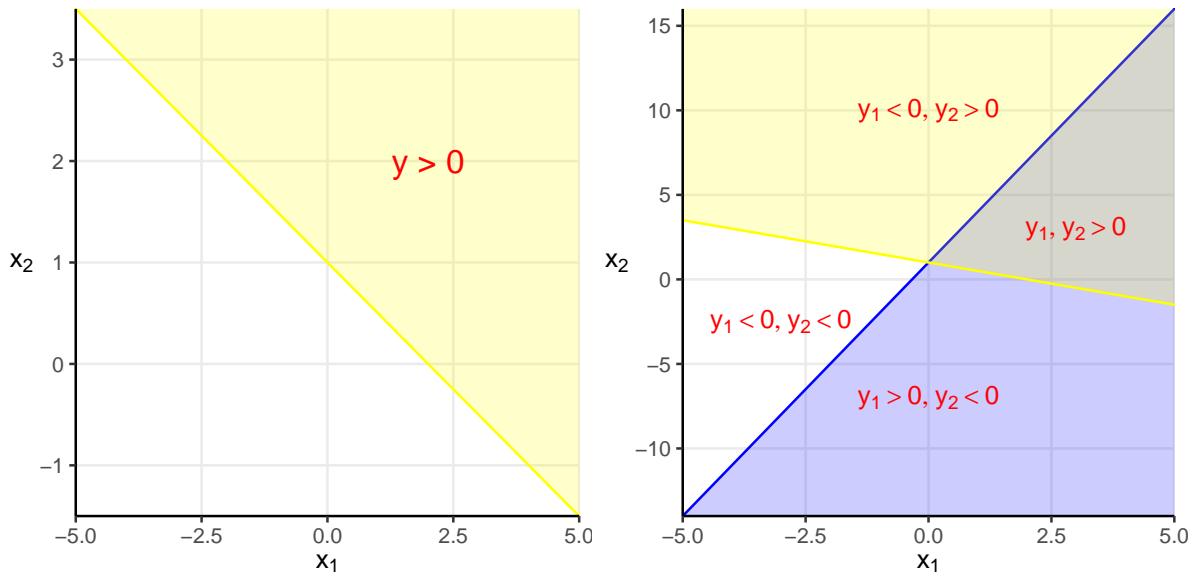


Figura 2: Hiperplano  $-2 + X_1 + 2X_2 = 0$  e interseções entre os planos

## Questão 2

We have seen that in  $p = 2$  dimensions, a linear decision boundary takes the form  $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$ . We now investigate a non-linear decision boundary.

### Item a)

Sketch the curve  $(1 + X_1)^2 + (2X_2)^2 = 4$

---

A Figura 3 mostra a elipse definida pela equação  $(1 + X_1)^2 + (2X_2)^2 = 4$ .

```
x1 <- seq(-4, 2, length.out = 100)
x2 <- seq(-2, 6, length.out = 100)

grid <- expand.grid(X1 = x1, X2 = x2)

grid$Z <- (1 + grid$X1)^2 + (2 - grid$X2)^2

ggplot(grid, aes(x = X1, y = X2, z = Z)) +
  geom_contour(aes(z = Z), breaks = 4, color = "black", linewidth = 1) +
  labs(
    x = "X1", y = "X2") +
  expand_limits(x = c(-4, 2), y = c(-2, 6)) +
  theme_minimal()
```



Figura 3

**Item b)**

On your sketch, indicate the set of points for which  $(1 + X_1)^2 + (2X_2)^2 > 4$ , as well as the set of points for which  $(1 + X_1)^2 + (2X_2)^2 \leq 4$ .

---

```
ggplot(subset(grid, Z > 4), aes(x = X1, y = X2)) +
  geom_point(color = "red", alpha = 0.2) +
  geom_point(data = subset(grid, Z <= 4), aes(x = X1, y = X2), color = "blue", alpha
             = 0.2) +
  labs(
    x = "X1", y = "X2") +
  theme_minimal()
```

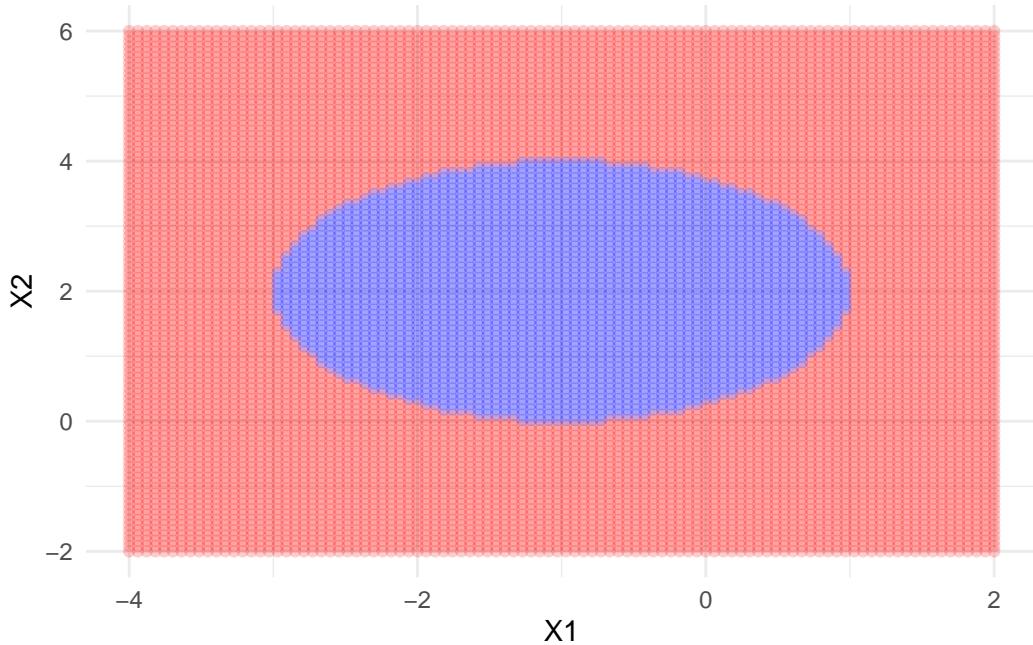


Figura 4

### Item c)

Suppose that a classifier assigns an observation to the blue class if  $(1 + X_1)^2 + (2X_2)^2 > 4$ , and to the red class otherwise. To what class is the observation  $(0, 0)$  classified?  $(-1, 1)$ ?  $(2, 2)$ ?  $(3, 8)$ ?

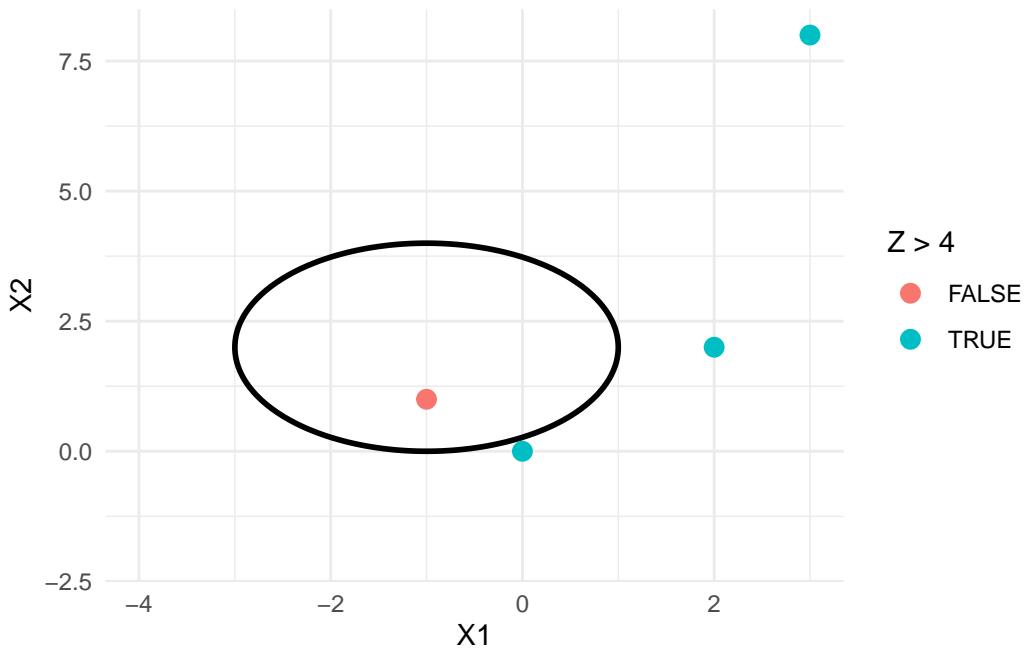
---

Em ordem, teríamos as classificações: azul, vermelho, azul, azul.

```
theta <- seq(0, 2*pi, length.out = 100)

pontos <- tibble(
  x = c(0, -1, 2, 3),
  y = c(0, 1, 2, 8),
  z = (1+x)^2 + (2-y)^2,
  cor = z > 4
)

ggplot(grid, aes(x = X1, y = X2, z = Z)) +
  geom_contour(aes(z = Z), breaks = 4, color = "black", linewidth = 1) +
  geom_point(data = pontos, aes(x = x, y = y, z = z, color = cor), size = 3) +
  labs(color = "Z > 4",
       x = "X1", y = "X2") +
  expand_limits(x = c(-4, 2), y = c(-2, 6)) +
  theme_minimal()
```



#### Item d)

Argue that while the decision boundary in (c) is not linear in terms of  $X_1$  and  $X_2$ , it is linear in terms of  $X_1$ ,  $X_1^2$ ,  $X_2$ , and  $X_2^2$ .

---

De fato, se expandirmos a equação, temos

$$(1 + X_1)^2 + (2 - X_2)^2 = 1 + 2X_1 + X_1^2 + 4 - 4X_2 + X_2^2 = 4X_1^2 + X_2^2 + 2X_1 - 4X_2 + 1 = 0,$$

que é linear com respeito a  $X_1$ ,  $X_1^2$ ,  $X_2$ , e  $X_2^2$ .

### Questão 3

Here we explore the maximal margin classifier on a toy data set.

#### Item a)

We are given  $n = 7$  observations in  $p = 2$  dimensions. For each observation, there is an associated class label. Sketch the observations.

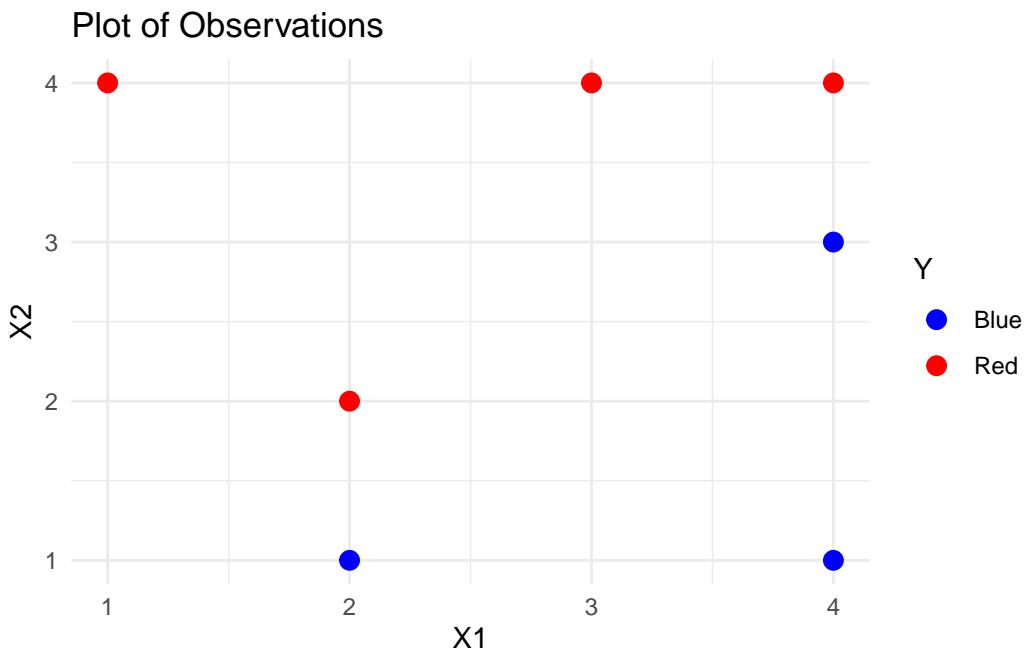
---

Criando o banco de dados do exercicio e fazendo um gráfico dele.

```

df <- data.frame(
  Obs = 1:7,
  X1 = c(3, 2, 4, 1, 2, 4, 4),
  X2 = c(4, 2, 4, 4, 1, 3, 1),
  Y = c("Red", "Red", "Red", "Red", "Blue", "Blue", "Blue")
)
ggplot(df, aes(x = X1, y = X2, color = Y)) +
  geom_point(size = 3) +
  scale_color_manual(values = c("Red" = "red", "Blue" = "blue")) +
  labs(x = "X1", y = "X2", title = "Plot of Observations") +
  theme_minimal()

```



### Item b)

Sketch the optimal separating hyperplane, and provide the equation for this hyperplane (of the form (9.1)).

Os pontos  $(x_1, x_2) = (2, 2)$  e  $(4, 3)$  parecem ser 2 bons pontos para se fazer hiperplano ótimo de separação.

```

# Calcular os coeficientes
ponto1 <- c(2, 2)
ponto2 <- c(4, 3)

# Inclinação da reta
m <- (ponto2[2] - ponto1[2]) / (ponto2[1] - ponto1[1])

# intercept

```

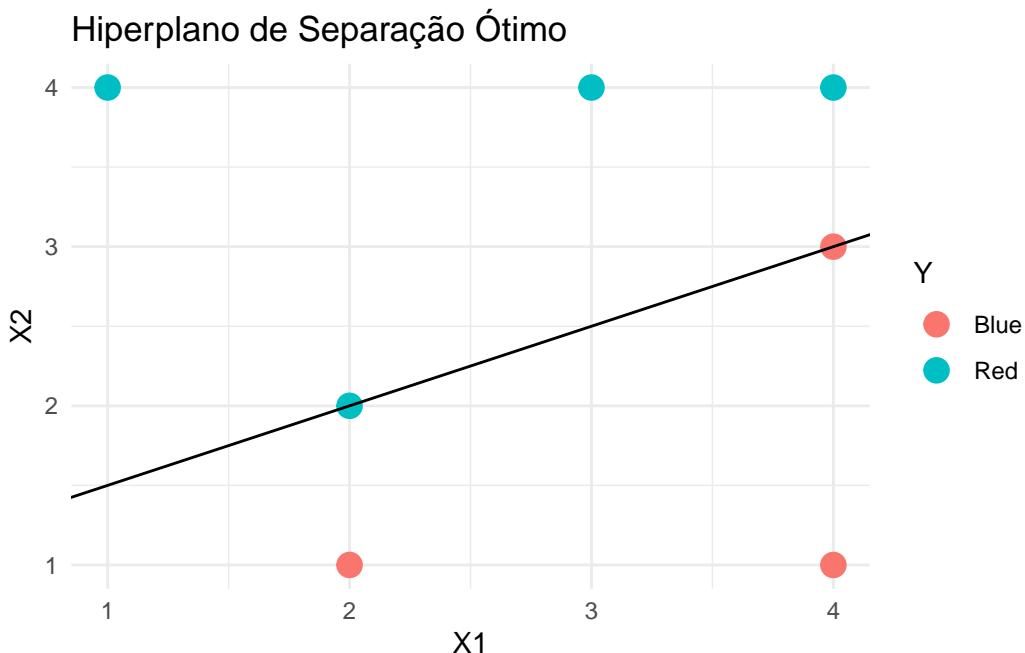
```

b <- ponto1[2] - m * ponto1[1]

# Equação da linha: X2 = m * X1 + b
# Converter para a forma: beta_1 * X1 + beta_2 * X2 + beta_0 = 0
beta1 <- -m
beta2 <- 1
beta0 <- -b

# Plotar com o hiperplano
ggplot(df, aes(x = X1, y = X2, color = Y)) +
  geom_point(size = 4) +
  geom_abline(intercept = b, slope = m, color = "black") +
  labs(title = "Hiperplano de Separação Ótimo") +
  theme_minimal()

```



```
cat("Equação do hiperplano: ")
```

Equação do hiperplano:

```
cat(paste0(beta0, " + ", beta1, " * X1 + ", beta2, " * X2 = 0"))
```

```
-1 + -0.5 * X1 + 1 * X2 = 0
```

### Item c)

Describe the classification rule for the maximal margin classifier. It should be something along the lines of “Classify to Red if  $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$ , and classify to Blue otherwise.” Provide the values for  $\beta_0$ ,  $\beta_1$ , and  $\beta_2$ .

```

cat("Se ")

Se
cat(paste0(beta0, " + ", beta1, " * X1 + ", beta2, " * X2 > 0"), "classifique como
→ vermelho e azul caso contrario")
-1 + -0.5 * X1 + 1 * X2 > 0 classifique como vermelho e azul caso contrario

```

### Item d)

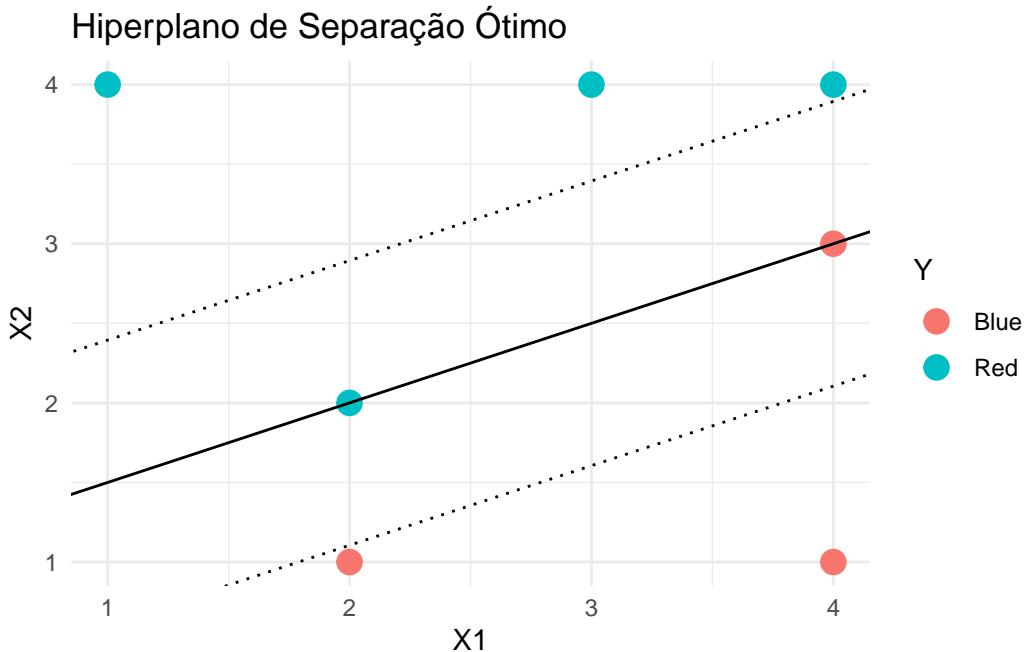
On your sketch, indicate the margin for the maximal margin hyperplane.

```

# Calcular a margem
margem <- 1 / sqrt(beta1^2 + beta2^2)

# Adicionar linhas de margem
ggplot(df, aes(x = X1, y = X2, color = Y)) +
  geom_point(size = 4) +
  geom_abline(intercept = b, slope = m, color = "black") +
  labs(title = "Hiperplano de Separação Ótimo") +
  theme_minimal()+
  geom_abline(intercept = (b + margem / beta2), slope = m, linetype = "dotted") +
  geom_abline(intercept = (b - margem / beta2), slope = m, linetype = "dotted")

```



**Item e)**

Indicate the support vectors for the maximal margin classifier.

---

Os pontos  $(x_1, x_2) = (2, 2)$  e  $(4, 3)$  foram usados como vetores de suporte.

**Item f)**

Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

---

Um movimento pequeno no ponto  $(4, 1)$  não afetaria a margem máxima do hiperplano visto que está distante da mesma e não é um dos vetores de suporte.

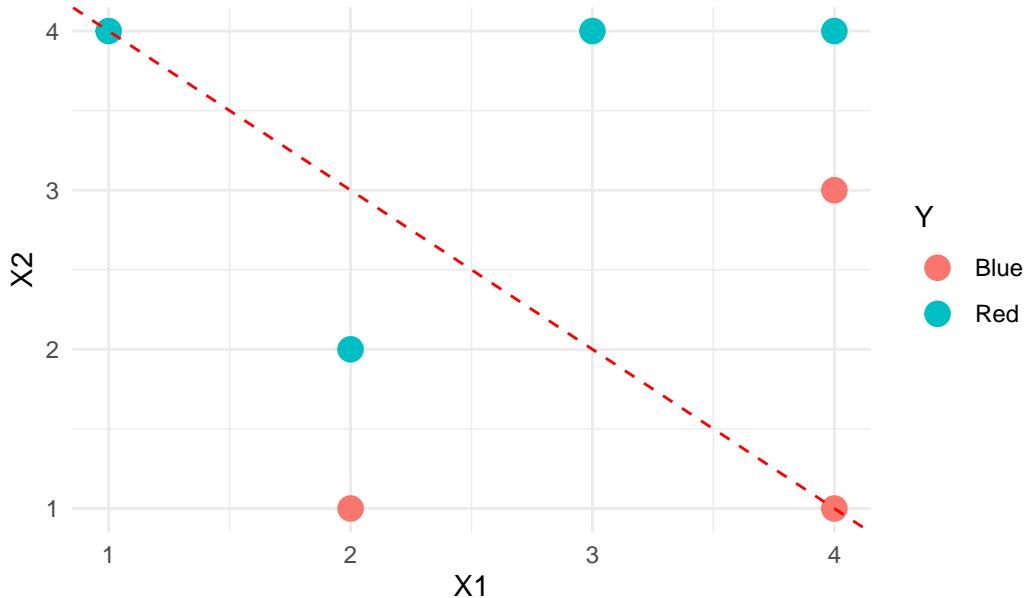
**Item g)**

Sketch a hyperplane that is not the optimal separating hyperplane, and provide the equation for this hyperplane.

---

```
# Hiperplano não ótimo
ggplot(df, aes(x = X1, y = X2, color = Y)) +
  geom_point(size = 4) +
  geom_abline(intercept = 5, slope = -1, color = "red", linetype = "dashed") +
  labs(title = "Hiperplano de Separação Não Ótimo") +
  theme_minimal()
```

## Hiperplano de Separação Não Ótimo



### Item h)

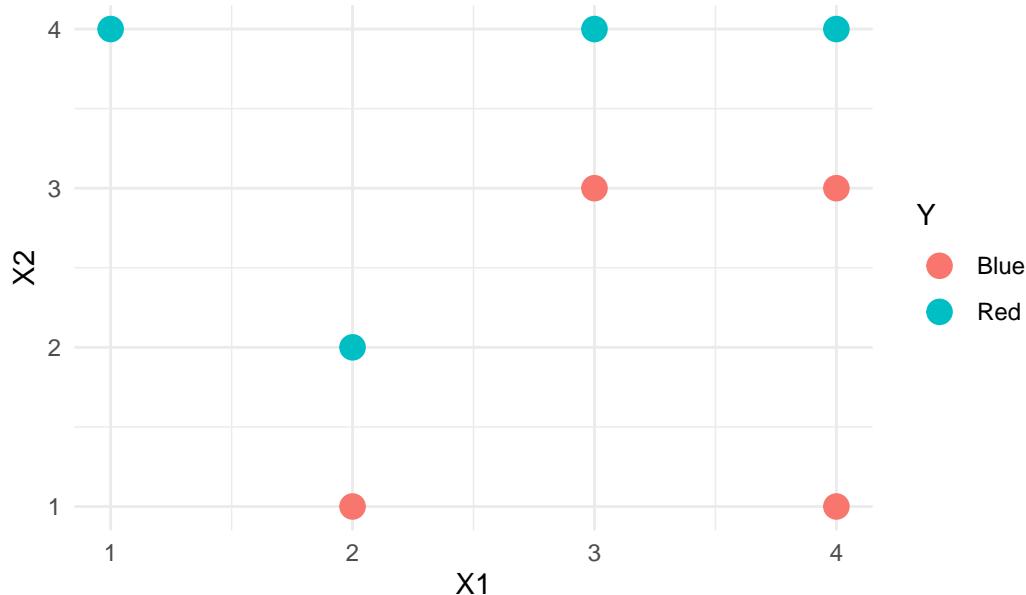
Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

---

```
# Adicionar nova observação
df2 <- rbind(df, data.frame(X1 = 3, X2 = 3, Y = as.factor('Blue'), obs=8))

# Plotar
ggplot(df2, aes(x = X1, y = X2, color = Y)) +
  geom_point(size = 4) +
  labs(title = "Classes não separaveis com uma nova observação") +
  theme_minimal()
```

## Classes não separáveis com uma nova observação



### Questão 4

Generate a simulated two-class data set with 100 observations and two features in which there is a visible but non-linear separation between the two classes. Show that in this setting, a support vector machine with a polynomial kernel (with degree greater than 1) or a radial kernel will outperform a support vector classifier on the training data. Which technique performs best on the test data? Make plots and report training and test error rates in order to back up your assertions.

---

Os dados são gerados a seguir, seguidos de um gráfico mostrando a separação entre as classes.

```
set.seed (1)
x <- matrix(rnorm (100 * 2), ncol = 2)
x[1:25, ] <- x[1:25, ] + 2
x[76:100, ] <- x[76:100, ] - 2
y <- c(rep(1, 25), rep(2, 50), rep(1, 25))
dat <- data.frame(x = x, y = as.factor(y))

plot(x, col = y)
```

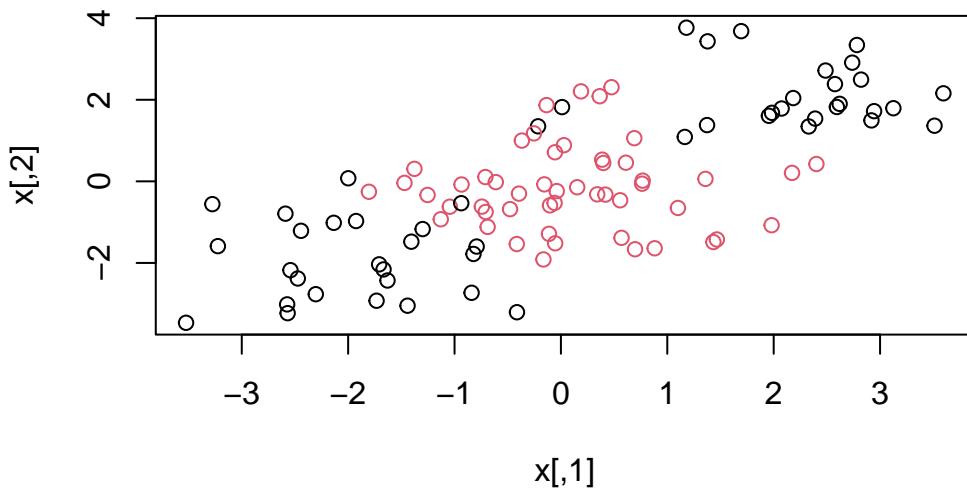


Figura 5

A seguir são ajustados os modelos SVM com kernel polinomial e radial, bem como o modelo SVM linear. Os resultados para uma partição de teste são apresentados na tabela a seguir utilizando custo igual a 1 em todos os casos, grau de polinômio igual a 2 no caso polinomial e  $\gamma$  igual a 1 no caso radial.

```
# seleciona parte da base como teste = 1
set.seed(1)
test <- sample(c(1, 0), 100, replace = TRUE, prob = c(0.2, 0.8))

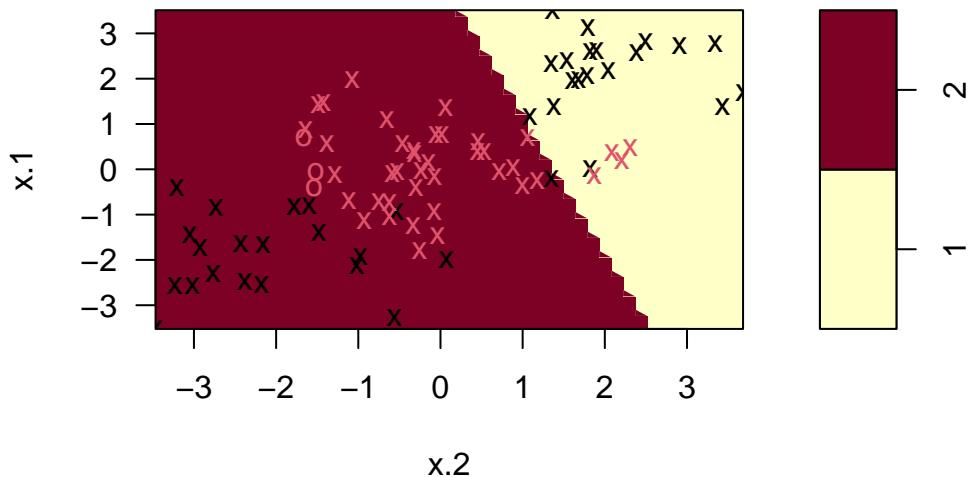
# classificador linear
linearfit <- svm(y ~ ., data = dat[test == 0,] , kernel = "linear",
                    cost = 1, scale = FALSE)

# classificador polinomial
polyfit <- svm(y ~ ., data = dat[test == 0,] , kernel = "polynomial",
                  degree = 2, cost = 1, scale = FALSE)

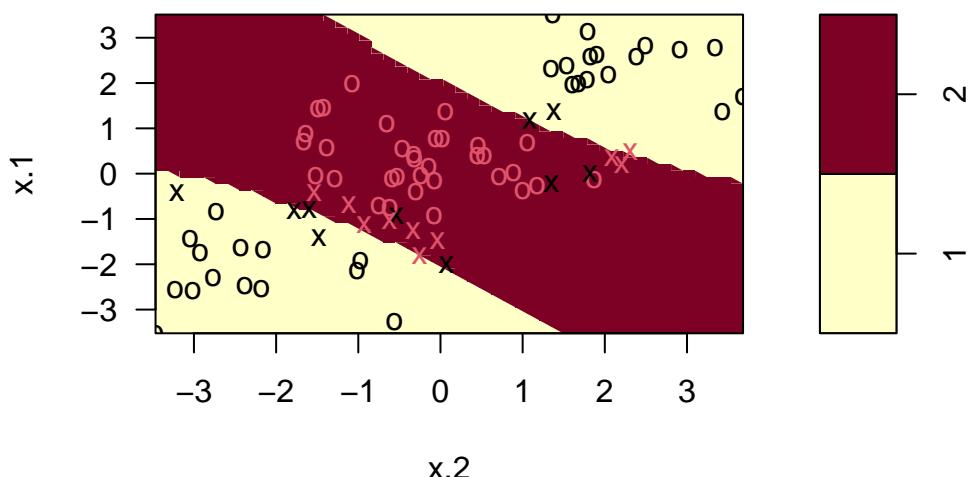
# classificador radial
radialfit <- svm(y ~ ., data = dat[test == 0,] , kernel = "radial",
                     gamma = 1, cost = 1, scale = FALSE)

# plots utilizando dados de treinamento
plot(linearfit, dat[test == 0,])
```

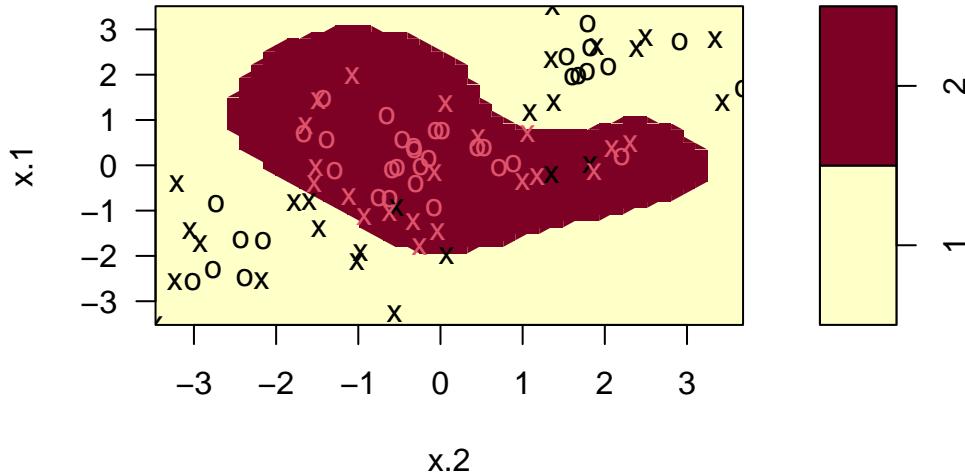
### SVM classification plot



### SVM classification plot



## SVM classification plot



Visualmente, o modelo SVM com kernel polinomial parece ser o que melhor se ajusta aos dados, contando apenas os vetores próximos à fronteira, como esperado, como vetores de suporte. A tabela a seguir mostra os resultados para os modelos ajustados com os dados de treinamento.

```
# table for linear fit
tbl_linear <- table(
  true = dat[test == 0, "y"],
  pred = predict(
    linearfit , newdata = dat[test == 0, ]
  )
)

# table for poly fit
tbl_poly <- table(
  true = dat[test == 0, "y"],
  pred = predict(
    polyfit , newdata = dat[test == 0, ]
  )
)

# table for radial fit
tbl_radial <- table(
  true = dat[test == 0, "y"],
  pred = predict(
    radialfit , newdata = dat[test == 0, ]
  )
)

map(list(tbl_linear, tbl_poly, tbl_radial), function(x) {
  x %>%
    bind_rows()
})
```

```

as.data.frame() %>%
  rename("True" = "true", "Predicted" = "pred") %>%
  mutate_all(as.character) %>%
  mutate(True = if_else(True == "1", "Classe real 1", "Classe real 2"),
         Predicted = if_else(Predicted == "1", "Classe fit 1", "Classe fit 2"))
  %>%
  pivot_wider(names_from = "True", values_from = "Freq")
}) %>%
  bind_rows() %>%
  mutate(Modelo = rep(c("Linear", "Polinomial", "Radial"), each = 2)) %>%
  select(Modelo, everything()) %>%
  knitr::kable(caption = "Matriz de confusão para os modelos ajustados com os dados
  de treinamento")

```

Tabela 1: Matriz de confusão para os modelos ajustados com os dados de treinamento

Modelo	Predicted	Classe real 1	Classe real 2
Linear	Classe fit 1	19	4
Linear	Classe fit 2	21	39
Polinomial	Classe fit 1	34	0
Polinomial	Classe fit 2	6	43
Radial	Classe fit 1	36	0
Radial	Classe fit 2	4	43

Novamente, a hipótese de que o kernel radial se ajusta melhor é suportada, visto que errou apenas 4 das 83 observações na base de treinamento, que corresponde a aproximadamente 5% de erro. O modelo polinomial teve desempenho marginalmente inferior, com aproximadamente 7% de erro.

A seguir é exposta a tabela com os resultados para os modelos ajustados com os dados de teste. Novamente, o desempenho do kerner radial é superior, com aproximadamente 5% de erro — 1 das 17 observações reservadas para teste.

```

# table for linear fit
tbl_linear <- table(
  true = dat[test == 1, "y"],
  pred = predict(
    linearfit , newdata = dat[test == 1, ]
  )
)

# table for poly fit
tbl_poly <- table(
  true = dat[test == 1, "y"],
  pred = predict(
    polyfit , newdata = dat[test == 1, ]
  )
)

```

```

# table for radial fit
tbl_radial <- table(
  true = dat[test == 1, "y"],
  pred = predict(
    radialfit , newdata = dat[test == 1, ]
  )
)

map(list(tbl_linear, tbl_poly, tbl_radial), function(x) {
  x %>%
    as.data.frame() %>%
    rename("True" = "true", "Predicted" = "pred") %>%
    mutate_all(as.character) %>%
    mutate(True = if_else(True == "1", "Classe real 1", "Classe real 2"),
           Predicted = if_else(Predicted == "1", "Classe fit 1", "Classe fit 2"))
    %>%
    pivot_wider(names_from = "True", values_from = "Freq")
}) %>%
bind_rows() %>%
mutate(Modelo = rep(c("Linear", "Polinomial", "Radial"), each = 2)) %>%
select(Modelo, everything()) %>%
knitr::kable(caption = "Matriz de confusão para os modelos ajustados com os dados
de teste")

```

Tabela 2: Matriz de confusão para os modelos ajustados com os dados de teste

Modelo	Predicted	Classe real 1	Classe real 2
Linear	Classe fit 1	5	0
Linear	Classe fit 2	5	7
Polinomial	Classe fit 1	10	2
Polinomial	Classe fit 2	0	5
Radial	Classe fit 1	10	1
Radial	Classe fit 2	0	6

## Questão 6

At the end of Section 9.6.1, it is claimed that in the case of data that is just barely linearly separable, a support vector classifier with a small value of `cost` that misclassifies a couple of training observations may perform better on test data than one with a huge value of `cost` that does not misclassify any training observations. You will now investigate this claim.

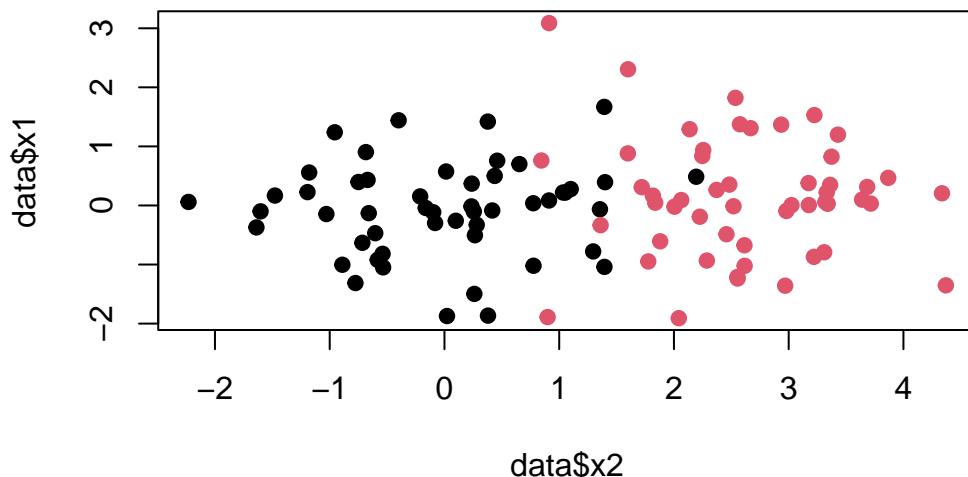
### Item a)

Generate two-class data with  $p = 2$  in such a way that the classes are just barely linearly separable.

Simulando 102 pontos de uma distribuição normal e adicionando um ruido de separação das classes para usarmos como um dado a ser classificado no exemplo.

```
set.seed(200)
data <- data.frame(
  x1 = rnorm(102),
  x2 = rnorm(102),
  y = factor(rep(c('A', 'B'), each=51))
)

# Adicionar uma pequena separação entre as classes
data$x2[51:102] <- data$x2[51:102] +2.6
plot(data$x2, data$x1, col=data$y, pch=19)
```



### Item b)

Compute the cross-validation error rates for support vector classifiers with a range of cost values. How many training observations are misclassified for each value of cost considered, and how does this relate to the cross-validation errors obtained?

---

Computando os erros de validação cruzada para os valores de custo 0.1, 1, 10 e 100. Pela utilização da função tune() o melhor valor de custo é de 100 visto que é o que tem o menor erro de validação cruzada.

```
tune.out <- tune(svm ,y ~ . , data = data , kernel = "linear",
                  ranges = list(cost = c( 0.1, 1, 10, 100,150)),scale=FALSE)
summary(tune.out)
```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

```

- best parameters:
cost
10

- best performance: 0.05909091

- Detailed performance results:
  cost      error dispersion
1  0.1 0.09818182 0.09229250
2  1.0 0.06909091 0.08200876
3 10.0 0.05909091 0.08389617
4 100.0 0.05909091 0.08389617
5 150.0 0.05909091 0.08389617

```

A seguir seram ajustados modelos para cada valor de custo e a matriz de confusão para cada modelo nos dados de treino.

### valor 0.1

Errou 8 observações.

```

modkernel1 <- svm(y ~ ., data=data, kernel="linear",cost=0.1)

pred1<-predict(modkernel1,data)
confusionMatrix(pred1,data$y)

```

Confusion Matrix and Statistics

		Reference	
Prediction	A	B	
A	47	4	
B	4	47	
			Accuracy : 0.9216
			95% CI : (0.8513, 0.9655)
			No Information Rate : 0.5
			P-Value [Acc > NIR] : <2e-16
			Kappa : 0.8431
			McNemar's Test P-Value : 1
			Sensitivity : 0.9216
			Specificity : 0.9216
			Pos Pred Value : 0.9216
			Neg Pred Value : 0.9216
			Prevalence : 0.5000
			Detection Rate : 0.4608
			Detection Prevalence : 0.5000
			Balanced Accuracy : 0.9216

```
'Positive' Class : A
```

### valor 1

Errou 7 observações.

```
modkernel2 <- svm(y ~ ., data=data, kernel="linear",cost=1)

pred2<-predict(modkernel2,data)
confusionMatrix(pred2,data$y)
```

Confusion Matrix and Statistics

Reference

Prediction	A	B
A	48	4
B	3	47

Accuracy : 0.9314  
95% CI : (0.8637, 0.972)  
No Information Rate : 0.5  
P-Value [Acc > NIR] : <2e-16

Kappa : 0.8627

McNemar's Test P-Value : 1

Sensitivity : 0.9412  
Specificity : 0.9216  
Pos Pred Value : 0.9231  
Neg Pred Value : 0.9400  
Prevalence : 0.5000  
Detection Rate : 0.4706  
Detection Prevalence : 0.5098  
Balanced Accuracy : 0.9314

```
'Positive' Class : A
```

### valor 10

Errou 7 observações.

```
modkernel3 <- svm(y ~ ., data=data, kernel="linear",cost=10)

pred3<-predict(modkernel3,data)
confusionMatrix(pred3,data$y)
```

Confusion Matrix and Statistics

Reference

```

Prediction A B
A 48 4
B 3 47

Accuracy : 0.9314
95% CI : (0.8637, 0.972)
No Information Rate : 0.5
P-Value [Acc > NIR] : <2e-16

Kappa : 0.8627

McNemar's Test P-Value : 1

Sensitivity : 0.9412
Specificity : 0.9216
Pos Pred Value : 0.9231
Neg Pred Value : 0.9400
Prevalence : 0.5000
Detection Rate : 0.4706
Detection Prevalence : 0.5098
Balanced Accuracy : 0.9314

'Positive' Class : A

```

## Valor 100

Errou 6 observações.

```

modkernel4 <- svm(y ~ ., data=data, kernel="linear", cost=100)

pred4<-predict(modkernel4,data)
confusionMatrix(pred4,data$y)

```

Confusion Matrix and Statistics

		Reference
Prediction	A	B
A	49	4
B	2	47

```

Accuracy : 0.9412
95% CI : (0.8764, 0.9781)
No Information Rate : 0.5
P-Value [Acc > NIR] : <2e-16

Kappa : 0.8824

```

McNemar's Test P-Value : 0.6831

Sensitivity : 0.9608

```

    Specificity : 0.9216
    Pos Pred Value : 0.9245
    Neg Pred Value : 0.9592
        Prevalence : 0.5000
    Detection Rate : 0.4804
Detection Prevalence : 0.5196
Balanced Accuracy : 0.9412

'Positive' Class : A

```

### Valor 150

```

modkernel5 <- svm(y ~ ., data=data, kernel="linear",cost=150)

pred5<-predict(modkernel5,data)
confusionMatrix(pred5,data$y)

```

Confusion Matrix and Statistics

		Reference	
		A	B
Prediction	A	49	4
	B	2	47
		Accuracy : 0.9412	
		95% CI : (0.8764, 0.9781)	
No Information Rate : 0.5			
P-Value [Acc > NIR] : <2e-16			
Kappa : 0.8824			

Mcnemar's Test P-Value : 0.6831

```

    Sensitivity : 0.9608
    Specificity : 0.9216
    Pos Pred Value : 0.9245
    Neg Pred Value : 0.9592
        Prevalence : 0.5000
    Detection Rate : 0.4804
Detection Prevalence : 0.5196
Balanced Accuracy : 0.9412

```

'Positive' Class : A

Como indicado na tabela de erros de validação cruzada o aumento de custo a partir do valor de custo 10, se torna redundante, classificando no máximo 1 observação a mais de forma certa.

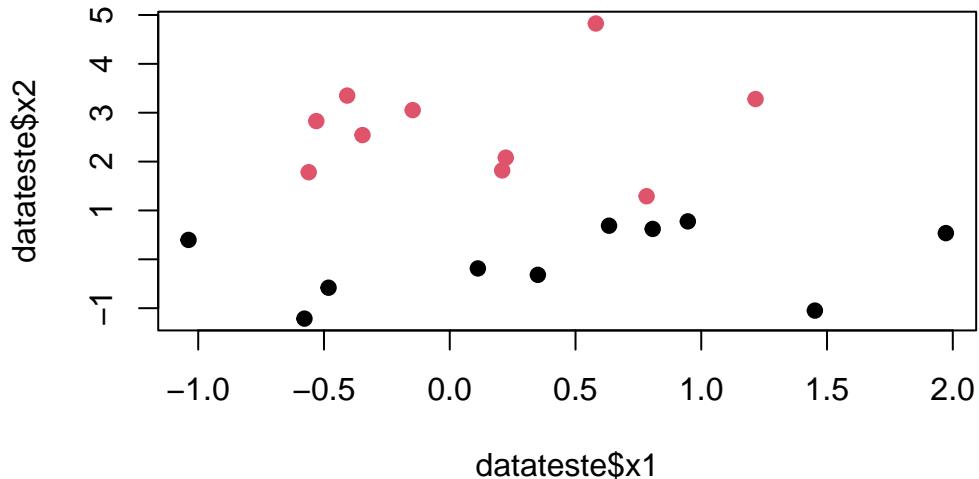
### Item c)

Generate an appropriate test data set, and compute the test errors corresponding to each of the values of cost considered. Which value of cost leads to the fewest test errors, and how does this compare to the values of cost that yield the fewest training errors and the fewest cross-validation errors?

---

Criando 20 amostras de teste que se asemelham aos dados de treinamento

```
datateste <- data.frame(  
  x1 = rnorm(20),  
  x2 = rnorm(20),  
  y = factor(rep(c('A', 'B'), each=10))  
)  
datateste$x2[11:20] <- datateste$x2[11:20]+2.2  
plot(datateste$x1, datateste$x2, col=datateste$y, pch=19)
```



A partir da função tune() é indicado que para os dados de teste o custo que melhor otimiza o modelo é 1

```
tune.out1 <- tune(svm ,y ~ . , data = data , kernel = "linear",  
                   ranges = list(cost = c( 0.1, 1, 10, 100,150)),scale=FALSE)  
summary(tune.out1)
```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation
- best parameters:
  - cost
  - 100
- best performance: 0.06727273

```

- Detailed performance results:
  cost      error dispersion
1 0.1 0.08636364 0.09312945
2 1.0 0.07727273 0.08476728
3 10.0 0.07727273 0.09699337
4 100.0 0.06727273 0.09951212
5 150.0 0.06727273 0.09951212

bestmodelteste<-tune.out1$best.model
predteste<-predict(bestmodelteste,datatest)
confusionMatrix(predteste,datatest$y)

```

Confusion Matrix and Statistics

		Reference
Prediction	A	B
A	10	1
B	0	9

Accuracy :	0.95
95% CI :	(0.7513, 0.9987)
No Information Rate :	0.5
P-Value [Acc > NIR] :	2.003e-05
 Kappa :	
0.9	
 McNemar's Test P-Value :	
1	
 Sensitivity :	
1.0000	
Specificity :	
0.9000	
Pos Pred Value :	
0.9091	
Neg Pred Value :	
1.0000	
Prevalence :	
0.5000	
Detection Rate :	
0.5000	
Detection Prevalence :	
0.5500	
Balanced Accuracy :	
0.9500	
 'Positive' Class :	
A	

#### Item d)

Discuss your results.

As questões acima mostram que a escolha do valor de custo deve ser feito com cuidado, de forma a não usar valores muito altos de custo ou seja mais restritos para a divisão de classes visto que a partir de um certo valor não ha mais ganho no aumento do valor do custo para a classificação do modelo.

## Questão 7

In this problem, you will use support vector approaches in order to predict whether a given car gets high or low gas mileage based on the `ISLR::Auto` data set.

### Item a)

Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

---

```
auto <- ISLR::Auto %>%
  mutate(mpg01 = if_else(mpg > median(mpg), 1, 0))
```

### Item b)

Fit a support vector classifier to the data with various values of `cost`, in order to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter. Comment on your results. Note you will need to fit the classifier without the gas mileage variable to produce sensible results.

---

A seguir é ajustado um classificador linear com valores de custo iguais a 0.1, 1, 10, 100 e 1000 e é usada validação cruzada com 10 folds para avaliar o desempenho do modelo. Isto é, a base de dados é particionada em 10 pedaços e o modelo é ajustado 10 vezes, cada vez utilizando 9 pedaços para treinamento e 1 para teste. O erro de classificação é calculado para cada ajuste e a média dos erros é reportada.

De acordo com os resultados do `summary()`, o modelo com custo igual a 1 apresentou os melhores resultados, com 9,5% de erro.

```
auto_nompq <- auto %>% select(-mpg)

set.seed (1)
tune.svc <- tune(svm, mpg01 ~ ., data = auto_nompq,
                  kernel = "linear",
                  ranges = list(
                    cost = c(0.1, 1, 10, 100, 1000)
                  )
)
summary(tune.svc)
```

Parameter tuning of 'svm':

```

- sampling method: 10-fold cross validation

- best parameters:
cost
  1

- best performance: 0.09603609

- Detailed performance results:
  cost      error dispersion
1 1e-01  0.10227373  0.03634911
2 1e+00  0.09603609  0.03666741
3 1e+01  0.10531309  0.03683207
4 1e+02  0.12079079  0.03864160
5 1e+03  0.12724775  0.03878303

```

### Item c)

Now repeat (b), this time using SVMs with radial and polynomial basis kernels, with different values of `gamma` and `degree` and `cost`. Comment on your results.

---

A seguir são ajustados modelos SVM com kernel polinomial e radial, variando os valores de `cost` e `degree` para o polinomial e `cost` e `gamma` para o radial.

```

set.seed (1)
tune.svpoly <- tune(svm, mpg01 ~ ., data = auto_nompg,
                      kernel = "polynomial",
                      ranges = list(
                        cost = c(0.1, 1, 10, 100, 1000),
                        degree = c(2, 3, 4, 5)
                      )
)

set.seed (1)
tune.svradial <- tune(svm, mpg01 ~ ., data = auto_nompg,
                      kernel = "radial",
                      ranges = list(
                        cost = c(0.1, 1, 10, 100, 1000),
                        gamma = c(0.5, 1, 2, 3, 4)
                      )
)

```

Os resultados para os modelos ajustados com kernel polinomial e radial são apresentados a seguir. O modelo polinomial via de regra apresenta resultados precários. O melhor resultado, com 15% de erros,

ocorre com um polinômio de grau 2 e erro igual a 1000. O modelo radial, por outro lado, apresenta resultados melhores, com erro de 6,5% para custo igual a 1 e  $\gamma$  igual a 0.5. Conclui-se que o modelo radial é o mais indicado, considerando apenas as simulações.

```
summary(tune.svpoly)
```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation
- best parameters:

cost	degree
1000	2
- best performance: 0.1594745
- Detailed performance results:

	cost	degree	error	dispersion
1	1e-01	2	0.4960905	0.03974005
2	1e+00	2	0.4752293	0.04525479
3	1e+01	2	0.3375643	0.08311313
4	1e+02	2	0.2548390	0.07343884
5	1e+03	2	0.1594745	0.05452093
6	1e-01	3	0.4978768	0.03935396
7	1e+00	3	0.4924827	0.03986701
8	1e+01	3	0.4418838	0.04913560
9	1e+02	3	0.2465708	0.05122197
10	1e+03	3	0.1683651	0.03763027
11	1e-01	4	0.4984602	0.03930559
12	1e+00	4	0.4983427	0.03933257
13	1e+01	4	0.4971505	0.03961089
14	1e+02	4	0.4855727	0.04370976
15	1e+03	4	0.3998639	0.09315902
16	1e-01	5	0.4984718	0.03930295
17	1e+00	5	0.4984585	0.03930611
18	1e+01	5	0.4983260	0.03933773
19	1e+02	5	0.4969797	0.03965863
20	1e+03	5	0.4838647	0.04389044

```
summary(tune.svradial)
```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation
- best parameters:

cost	gamma
1	0.5

```

- best performance: 0.0657775

- Detailed performance results:
  cost gamma    error dispersion
1 1e-01 0.5 0.08548406 0.021104890
2 1e+00 0.5 0.06577750 0.026197728
3 1e+01 0.5 0.07304343 0.027869128
4 1e+02 0.5 0.07667563 0.027824986
5 1e+03 0.5 0.07667563 0.027824986
6 1e-01 1.0 0.27951125 0.036343997
7 1e+00 1.0 0.09918732 0.020523476
8 1e+01 1.0 0.10442706 0.020690277
9 1e+02 1.0 0.10442540 0.020692046
10 1e+03 1.0 0.10442540 0.020692046
11 1e-01 2.0 0.42047779 0.044561140
12 1e+00 2.0 0.20409900 0.013765977
13 1e+01 2.0 0.20506214 0.013269495
14 1e+02 2.0 0.20506214 0.013269495
15 1e+03 2.0 0.20506214 0.013269495
16 1e-01 3.0 0.43453116 0.043273628
17 1e+00 3.0 0.23107129 0.010987668
18 1e+01 3.0 0.23126372 0.010794330
19 1e+02 3.0 0.23126372 0.010794330
20 1e+03 3.0 0.23126372 0.010794330
21 1e-01 4.0 0.43894907 0.041927995
22 1e+00 4.0 0.23592639 0.009193937
23 1e+01 4.0 0.23597971 0.009134275
24 1e+02 4.0 0.23597971 0.009134275
25 1e+03 4.0 0.23597971 0.009134275

```

#### Item d)

Make some plots to back up your assertions in (b) and (c).

---

A seguir, para comparar os modelos de forma sintética, são comparadas as curvas ROC para os melhores modelos de cada categoria. O que se pode observar na Figura 6 é uma confirmação de que o modelo polinomial tem um desempenho consideravelmente pior que os demais. Enquanto o modelo radial parece apresentar quase desempenho perfeito — o que pode ser um indicador de overfitting — o modelo linear apresenta um desempenho comparável a este e, por isso, pode apresentar uma flexibilidade maior para classificação de novas observações.

```

fit_svc <- attributes(predict(tune.svc$best.model , auto_nompg, decision.values =
  ~ T))$decision.values

fit_svpoly <- attributes(predict(tune.svpoly$best.model , auto_nompg,
  ~ decision.values = T))$decision.values

```

```

fit_svradial <- attributes(predict(tune.svradial$best.model , auto_nompg,
→ decision.values = T))$decision.values

rocplot(fit_svc , auto_nompg$mpg01, col = "red")
rocplot(fit_svpoly , auto_nompg$mpg01, add = T, col = "blue")
rocplot(fit_svradial , auto_nompg$mpg01, add = T, col = "green")
legend("bottomright", legend = c("Linear", "Poli.", "Radial"), col = c("red",
→ "blue", "green"), pch = 16)

```

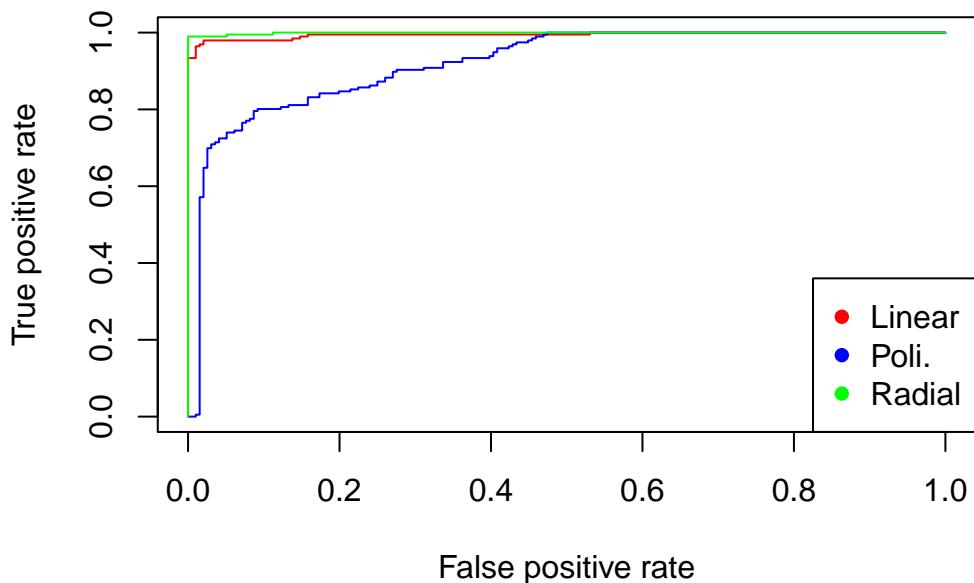


Figura 6: Curvas ROC para os modelos ajustados com SVMs

### Questão 13

Escolha uma linguagem de programação (R, Python, SAS, Matlab, Julia) e apresente um exemplo de classificação com SVM utilizando Kernel Linear e outro utilizando Kernel não Linear.

---

No SAS, a função implementada para realizar análises utilizando algoritmos de suporte vetorial (SVM) é o PROC HPSVM. Essa função permite o uso de kernels lineares e não lineares nos dados de treinamento. O PROC HPSVM executa o algoritmo SVM de alta performance, possibilitando sua execução em computação paralela tanto em uma única máquina quanto em múltiplas máquinas.<sup>1</sup>

Utilizando o exemplo fornecido na documentação do SAS com um kernel linear, vamos ajustar um modelo utilizando um banco de dados chamado SAMPSON.DMAGECR, um banco de dados de referência

---

<sup>1</sup>[https://documentation.sas.com/doc/en/ehpprcref/14.2/ehpprcref\\_hpsvm\\_overview.htm](https://documentation.sas.com/doc/en/ehpprcref/14.2/ehpprcref_hpsvm_overview.htm)

que traz informações sobre risco de crédito<sup>2</sup>. Esse banco de dados faz parte da biblioteca SAMPSON e contém 1.000 observações, cada uma com informações detalhadas sobre os requerentes. O banco inclui a classificação do indivíduo como GOOD ou BAD em uma variável denominada GOOD\_BAD, além de outras variáveis como histórico de crédito, duração do empréstimo, entre outras.

A tabela ‘Matriz de Classificação’ mostra que, entre as 1.000 observações totais, 700 são classificadas como boas e 300 como ruins. O número de observações BOAS corretamente previstas é 626, e o número de observações RUINS corretamente previstas é 158.

Classification Matrix			
Observed	Training Prediction		
	good	bad	Total
good	626	74	700
bad	142	158	300
Total	768	232	1000

Assim, a precisão do modelo é de 78,4%, conforme indicado na tabela ‘Estatísticas de Ajuste’.

Fit Statistics	
Statistic	Training
Accuracy	0.7840
Error	0.2160
Sensitivity	0.8943
Specificity	0.5267

Um modelo relativamente bom significa que a taxa de erro de classificação é baixa, enquanto a sensibilidade e a especificidade são altas. Com o PROC HPSVM, é possível ajustar os parâmetros de treinamento e usar diferentes kernels para obter um modelo melhor. O padrão do procedimento usa o kernel linear conforme especificado:

$$k(x_1, x_2) = \langle x_1, x_2 \rangle, \quad (1)$$

onde  $x_1$  e  $x_2$  são dois vetores e  $\langle \cdot, \cdot \rangle$  é o produto interno. Para alterar o tipo de kernel utilizado, basta definir o argumento KERNEL, especificando o tipo de kernel e quaisquer parâmetros associados. Definindo o kernel como polinomial:

$$k(x_1, x_2) = (\langle x_1, x_2 \rangle + 1)^p, \quad (2)$$

onde  $p$  é o grau do polinômio, obtém-se o resultado:

Classification Matrix			
Observed	Training Prediction		
	good	bad	Total
good	699	1	700
bad	3	297	300
Total	702	298	1000

A matriz de classificação mostra que o número de observações BOAS corretamente previstas é 699, e o número de observações RUINS corretamente previstas é 297. Assim, a precisão do modelo é de 99,6%, conforme indicado na tabela ‘Estatísticas de Ajuste’, mostrando que o uso do kernel polinomial é mais acurado, nesse caso.

---

<sup>2</sup><https://support.sas.com/documentation/cdl/en/emgs/59885/HTML/default/viewer.htm#a001026918.htm>

Fit Statistics	
Statistic	Training
Accuracy	0.9980
Error	0.0040
Sensitivity	0.9986
Specificity	0.9900

O código a seguir foi utilizado para gerar os resultados desta questão:

```

proc freq data = sampsio.dimagecr;
tables GOOD_BAD;
run;

proc hpsvm data=sampsio.dimagecr;
    input checking history purpose savings employed marital coapp
        property other job housing telephon foreign/level=nominal;
    input duration amount installp resident existcr depends age/level=interval;
    target good_bad;
    KERNEL LINEAR;
run;

/* Ajustando o modelo SVM
proc hpsvm data=sampsio.dimagecr;
    input checking history purpose savings employed marital coapp
        property other job housing telephon foreign / level=nominal;
    input duration amount installp resident existcr depends age / level=interval;
    target good_bad / level=nominal;
    id duration amount;
    savestate rstore=work.svm_model;
run;

Aplicando o modelo ajustado aos dados e salvando os resultados
proc astore;
    score data=sampsio.dimagecr out=svm_results rstore=work.svm_model;
run;

Gerando um gráfico de dispersão para visualizar a distribuição das classes previstas
→
proc sgplot data=svm_results;
    scatter x=amount y=duration / group=P_good_bad;
    xaxis label="Amount";
    yaxis label="Duration";
    title "Gráfico de Dispersão das Classes Previstas pelo Modelo SVM";
run; */

proc hpsvm data=sampsio.dimagecr;
    input checking history purpose savings employed marital coapp
        property other job housing telephon foreign/level=nominal;
    input duration amount installp resident existcr depends age/level=interval;
    target good_bad;

```

```
 KERNEL POLYNOM;  
run;
```

## Questão 5

We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

```
def plot_points(X1, X2, y, title, xlabel="X1", ylabel="X2", y2=None):
    fig, ax = plt.subplots(figsize=(8, 8))

    ax.set(title=title, xlabel=xlabel, ylabel=ylabel)

    scatter = ax.scatter(X1, X2, c=y, cmap=plt.cm.coolwarm)

    # Adicionar marcador 'x' nos pontos onde o predito é diferente do real
    if y2 is not None:
        for i in range(len(y)):
            if y[i] != y2[i]:
                ax.scatter(X1[i], X2[i], marker='x', color='black', s=100)

    plt.show()

def plot_clf(model, df, grid_range, show_contours=False, predicted=None):
    variables = df.columns

    # Prepara o grid de valores para plotagem
    xx = np.meshgrid(*[grid_range]*len(variables), sparse=False)
    Xgrid = np.vstack([xx[i].flatten() for i in range(len(xx))]).T
    x1 = grid_range
    x2 = grid_range
    xx1, xx2 = np.meshgrid(x1, x2, sparse=False)
    Xgrid = np.stack((xx1.flatten(), xx2.flatten())).T

    decision_boundary = model.predict(Xgrid)
    decision_boundary_grid = decision_boundary.reshape(len(x2), len(x1))

    plt.figure(figsize=(10, 10))
    if show_contours:
        plt.pcolormesh(xx1, xx2, decision_boundary_grid, alpha=0.3, cmap='coolwarm',
        → shading='auto')
        plt.contour(x1, x2, decision_boundary_grid)

    scatter = sns.scatterplot(x='X1', y='X2', hue='y', data=df, palette = {0:
    → 'blue', 1: 'red'}, edgecolor='w', s=100)
    scatter.legend_.remove()
    if predicted is not None:
        x_coords = df.loc[predicted != df['y'], 'X1']
        y_coords = df.loc[predicted != df['y'], 'X2']
        plt.scatter(x_coords, y_coords, color='black', marker='x', s=100,
        → label='Predicted')

    plt.show()
```

```
models = {}
```

Como preparação, foram criadas a função `plot_points`, a função `plot_clf` e o objeto `models`, que serão, respectivamente, uma função que facilita a elaboração dos gráficos, uma função que plota os limites de decisão e um objeto (dicionário) que armazena os modelos.

### Item a)

Generate a data set with  $n = 500$  and  $p = 2$ , such that the observations belong to two classes with a quadratic decision boundary between them. For instance, you can do this as follows:

```
rng = np.random.default_rng(5)

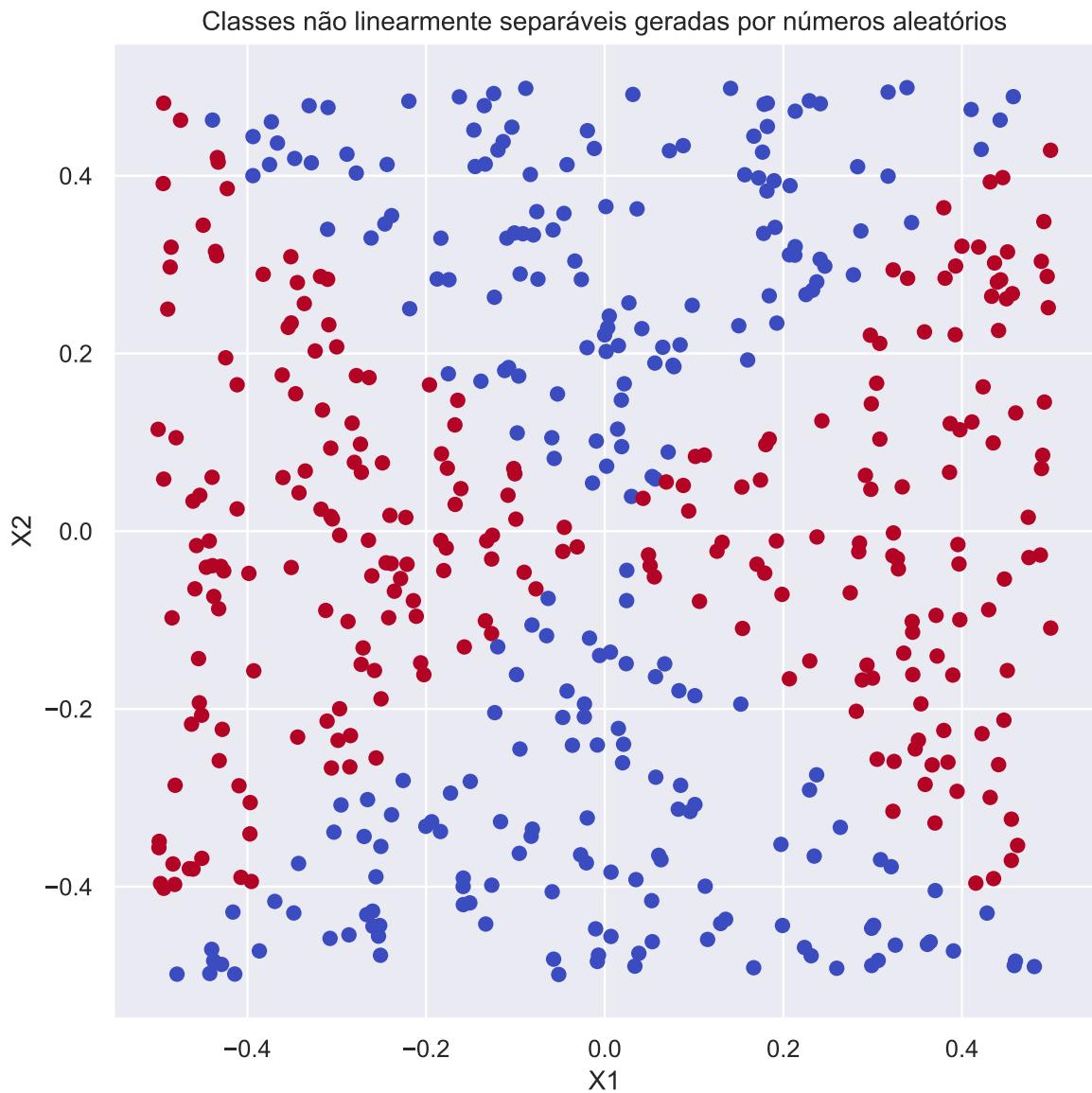
# Gerando valores de mesma distribuição
X = pd.DataFrame({
    "X1": rng.uniform(size=500) - 0.5,
    "X2": rng.uniform(size=500) - 0.5
})

# Criando uma categoria como função da diferença dos quadrados dos campos
y = (X['X1'] ** 2 - X['X2'] ** 2) > 0
```

### Item b)

Plot the observations, colored according to their class labels. Your plot should display  $X1$  on the x-axis, and  $X2$  on the y-axis.

```
plot_points(
    X['X1'], X['X2'], y,
    title='Classes não linearmente separáveis geradas por números aleatórios'
)
```



Apesar de haver uma clara separação entre os dados (inclusive de forma determinística, pois nenhuma classe se encontra na região visível da outra classe), não existe uma reta única capaz de separar os dados de forma satisfatória.

A seguir, serão testadas algumas separações.

### Item c)

Fit a logistic regression model to the data, using  $X_1$  and  $X_2$  as predictors.

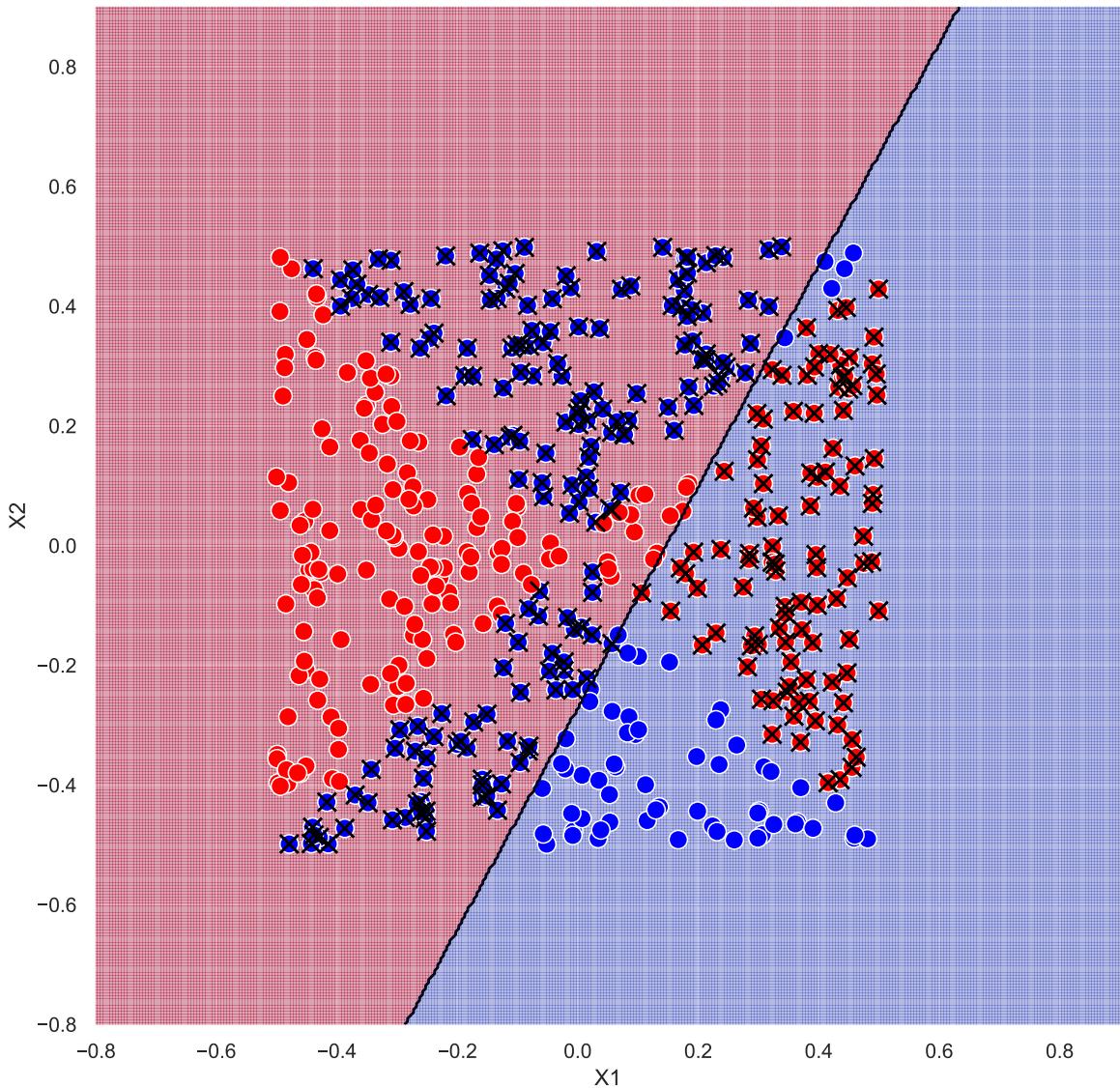
```
models.update(linlim = LogisticRegression().fit(X, y))
```

Com o modelo logístico ajustado, espera-se que a separação entre as categorias seja linear.

#### Item d)

Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be linear.

```
plot_clf(  
    models["linlim"], X.assign(y = y),  
    np.arange(-.8, .9, .005),  
    show_contours=True,  
    predicted=models["linlim"].predict(X)  
)
```



O modelo separa metade da região como uma das classes e metade como outra, através de uma reta diagonal.

Apesar disso, em ambas as classes, mais de metade da região correspondente a uma classe foi classificada como parte da outra. Esse modelo, portanto, não chega a ser melhor do que um lance de moeda, mesmo nos dados utilizados para o treinamento.

De fato, temos:

```
models["linlim"].score(X, y)
```

0.43

- e) Now fit a logistic regression model to the data using non-linear functions of  $X_1$  and  $X_2$  as predictors (e.g.  $X_1^2$ ,  $X_1 \times X_2$ ,  $\log(X_2)$ , and so forth).

Foram feitas duas escolhas de transformação:

- Quadrática;
- Exponencial;

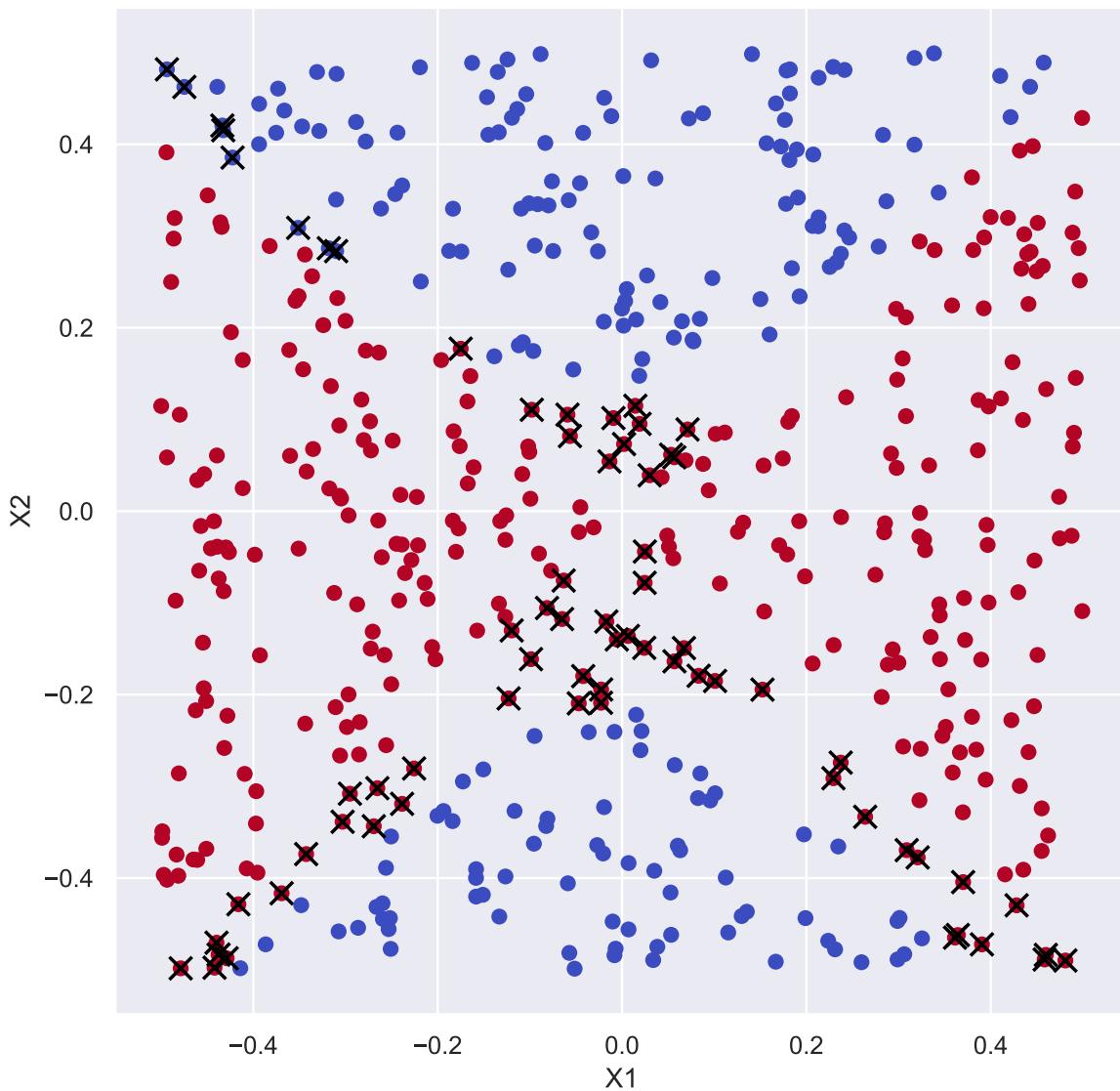
A escolha foi feita devido à presença de valores negativos em ambas as variáveis, o que torna funções como logaritmo e raízes quadradas inadequadas (gerariam valores inexistentes em  $\mathbb{R}$ ).

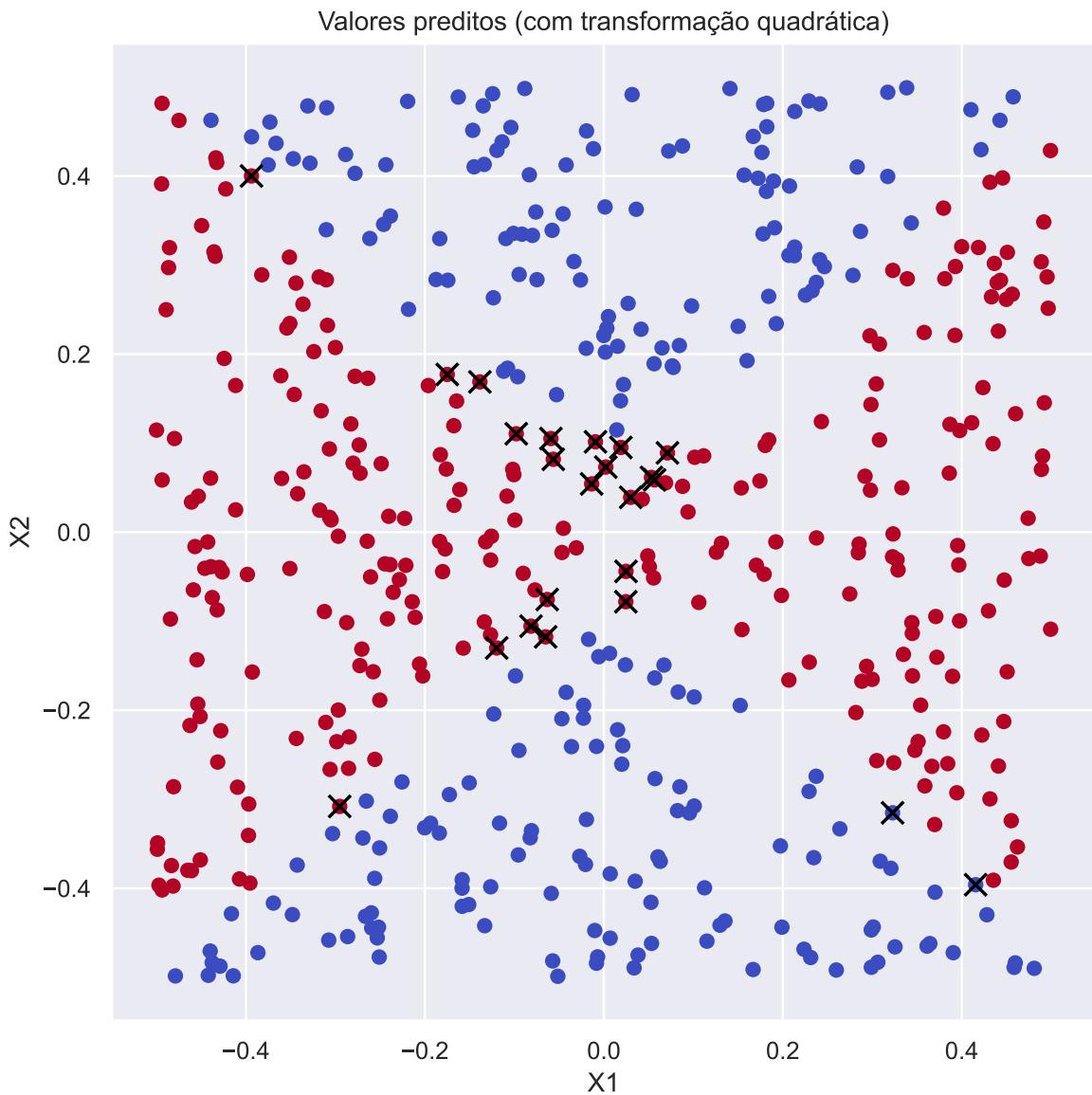
```
# criando um objeto (dicionário) que armazena dados para diferentes transformações
dmatrices = {
    "exponencial": X.assign(X1_exp = np.exp(X['X1']), X2_exp = np.exp(X['X2'])),
    "quadrática": X.assign(X1_2 = X['X1'] ** 2, X2_2 = X['X2'] ** 2)
}

# atualizando os modelos para essas previsões
models.update({
    mdl: LogisticRegression().fit(dmatrix, y)
    for mdl, dmatrix in dmatrices.items()
})

# fazendo gráfico para cada modelo
for transf in ("exponencial", "quadrática"):
    plot_points(
        X['X1'], X['X2'], models[transf].predict(dmatrices[transf]),
        title=f'Valores preditos (com transformação {transf})',
        y2 = y
    )
```

Valores preditos (com transformação exponencial)





Ambas as transformações tiveram separações muito melhores que a separação linear, como esperado.

```
for transf in ("exponencial", "quadrática"):
    print(f"Precisão do modelo com transformação {transf}:
          → {models[transf].score(dmatrices[transf], y)}")
```

Precisão do modelo com transformação exponencial: 0.862  
 Precisão do modelo com transformação quadrática: 0.954

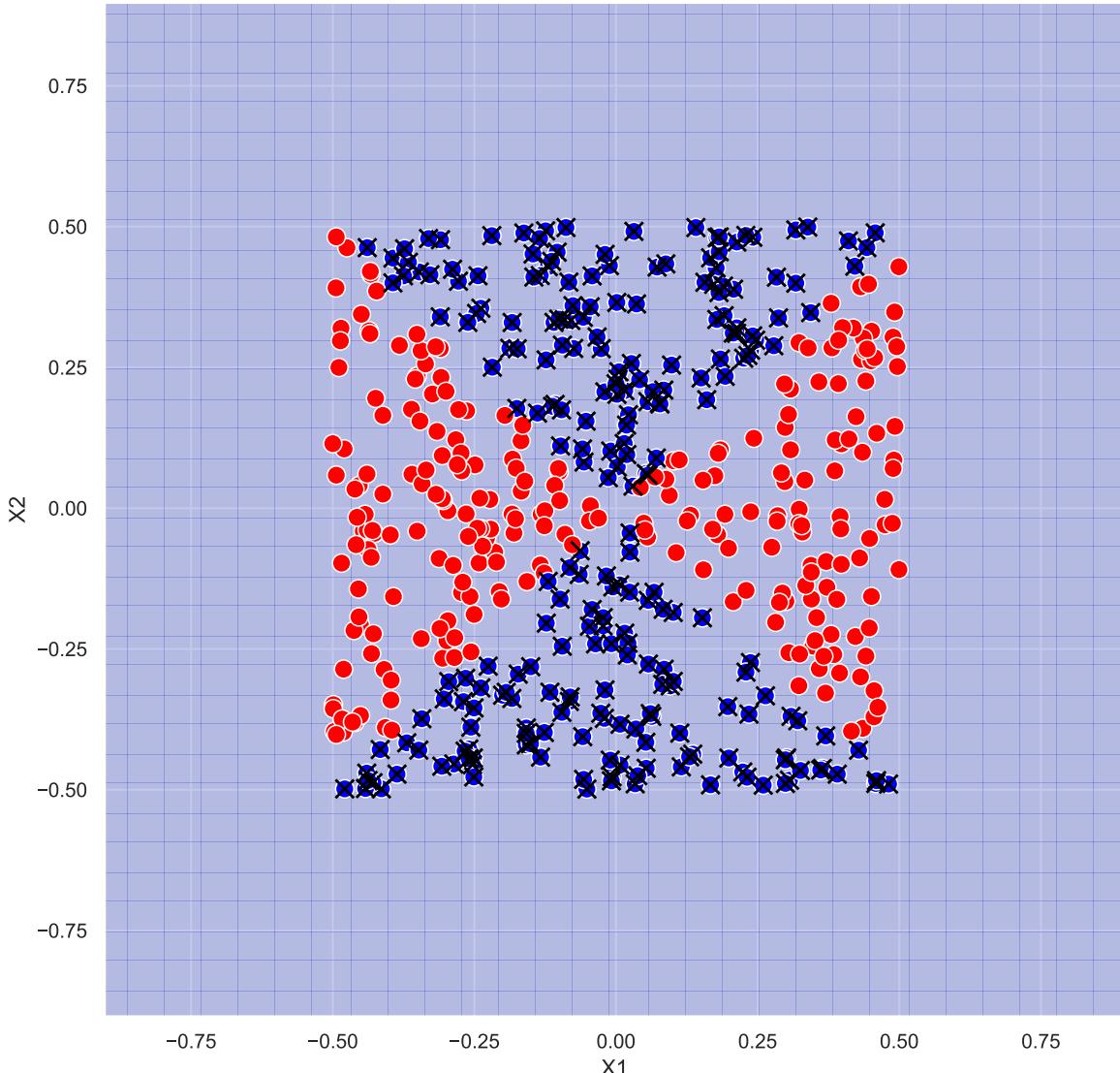
O modelo com transformações exponenciais errou mais que o modelo com transformações quadráticas, o que era esperado, dado que a variável y também foi gerada utilizando transformações quadráticas das variáveis. Este que, por sua vez, acertou em mais de 90% dos casos.

### Item g)

Fit a support vector classifier to the data with X1 and X2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
model_svm_linear = SVC(kernel='linear', C=1).fit(X, y)

plot_clf(model_svm_linear, X.assign(y = y), np.arange(-.9, .9, .005),
         show_contours=True, predicted=model_svm_linear.predict(X))
```



O modelo obteve um comportamento único: Todas as observações foram classificadas na mesma categoria, de forma que ele acertou todas de uma e errou todas de outra, atingindo uma acurácia de 50%.

```
model_svm_linear.score(X, y)
```

0.506

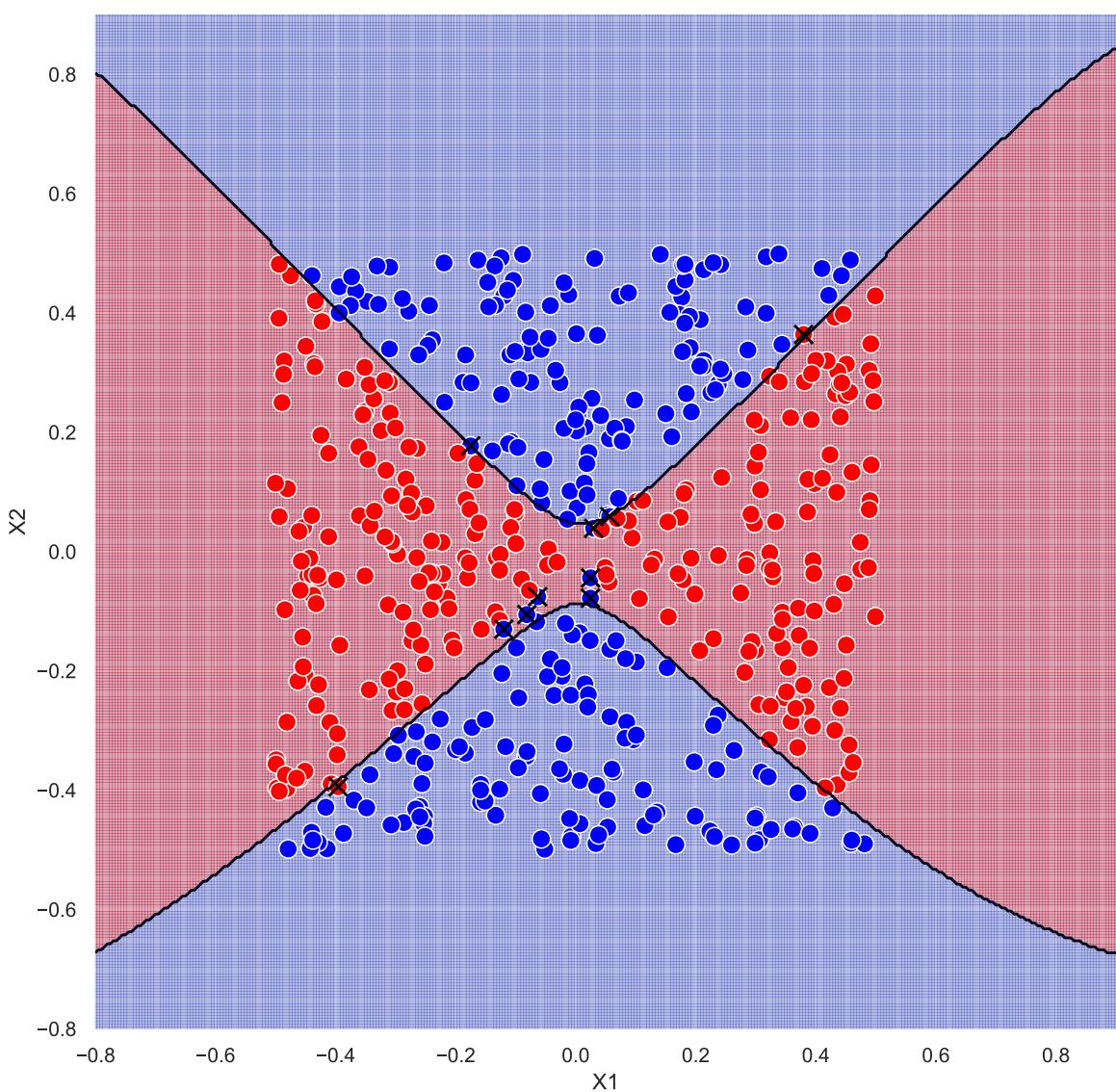
De fato, não era esperado que o modelo atingisse uma acurácia satisfatória.

### Item h)

Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
model_rbf = SVC(kernel='rbf', gamma=2, C=1, random_state=0, probability=True).fit(X,
→ y)

plot_clf(model_rbf, X.assign(y = y), np.arange(-.8, .9, .005), show_contours=True,
→ predicted=model_rbf.predict(X))
```



O modelo com kernel radial obteve um desempenho muito próximo dos valores reais.

```
model_rbf.score(X, y)
```

0.98

De fato, de todos os modelos testados, o SVM com kernel radial obteve a maior de todas as acuráncias.

### Item i)

Comment on your results.

Em todas as tentativas de elaborar um modelo que separasse linearmente os resultados, a acurácia não foi melhor que um chute, de modo que esses modelos não são úteis na aplicação desse caso.

Os modelos que consideravam não-linearidade tiveram, todos, um desempenho maior que 80% (no quesito acurácia), mesmo quando a função de transformação foi exponencial (que é diferente da função geradora original, que considerava transformações quadráticas).

Apesar de um dos modelos logísticos ter utilizado uma função quadrática como transformação, ele ainda obteve precisão menor que o SVM com kernel radial, que acertou a classificação de quase todas as observações.

A execução também permite observar diferenças de linguagem. Enquanto boa parte os gráficos já estão pré-implementados em R, as operações gráficas precisam ser feitas manualmente em Python, mesmo carregando as bibliotecas.

As fronteiras de decisão, quando testados em modelos logísticos com transformações, geraram objetos de memória com vários terabytes (que inviabilizou a exibição das superfícies), comportamento que não foi observado na construção em R. Isso mostra uma desvantagem do Python em relação ao R, que é generalizada, uma vez que o R consegue um desempenho satisfatório na visualização mesmo utilizando de funções padrão.

Vale comentar que a biblioteca ISLP, em Python, tem um método (`ISLP.svm.plot`) que permite traçar a superfície de decisão, recebendo um modelo e um conjunto de dados como argumentos. Porém, essa aplicação só vale para modelos SVM, diferente do R, onde o método “plot” se aplica a uma vasta gama, não só de modelos, como de tipos de dados diferentes.

## Questão 8

This problem involves the OJ data set which is part of the ISLP package.

```
def glimpse(df):
    print(f"Rows: {df.shape[0]}")
    print(f"Columns: {df.shape[1]}")
    for col in df.columns:
        print(f"${col} <{df[col].dtype}>\n→ {df[col].drop_duplicates().head().values}")

OJ = load_data('OJ')

glimpse(OJ)
```

```
Rows: 1070
Columns: 18
$ Purchase <category> ['CH', 'MM']
Categories (2, object): ['CH', 'MM']
$ WeekofPurchase <int64> [237 239 245 227 228]
$ StoreID <int64> [1 7 2 3 4]
$ PriceCH <float64> [1.75 1.86 1.69 1.99 2.06]
$ PriceMM <float64> [1.99 2.09 1.69 2.13 2.18]
$ DiscCH <float64> [0. 0.17 0.27 0.37 0.47]
$ DiscMM <float64> [0. 0.3 0.4 0.54 0.8 ]
$ SpecialCH <int64> [0 1]
$ SpecialMM <int64> [0 1]
$ LoyalCH <float64> [0.5 0.6 0.68 0.4 0.956535]
$ SalePriceMM <float64> [1.99 1.69 2.09 1.59 2.13]
$ SalePriceCH <float64> [1.75 1.69 1.86 1.59 1.99]
$ PriceDiff <float64> [ 0.24 -0.06 0.4 0. 0.3 ]
$ Store7 <category> ['No', 'Yes']
Categories (2, object): ['No', 'Yes']
$ PctDiscMM <float64> [0. 0.150754 0.201005 0.253521 0.366972]
$ PctDiscCH <float64> [0. 0.091398 0.145161 0.198925 0.252688]
$ ListPriceDiff <float64> [0.24 0.23 0. 0.3 0.27]
$ STORE <int64> [1 0 2 3 4]
```

Os dados possuem um número de dimensões maior do que é possível ser exibido em um gráfico, e tem dados de compradores de um suco de laranja.

Uma das colunas, Store7, é uma coluna indicadora (“Sim”, “Não”), que não está em formato numérico. Para evitar problemas na modelagem, o dado será convertido.

A outra coluna, Purchase, que indica se o cliente comprou Citrus Hill ou Minute Maid, será a coluna de resposta, de modo que não precisa ser convertida, portanto.

```
OJ = OJ.assign(
    Store7 = lambda x: x.Store7.apply(
        lambda y: {"No": 0, "Yes": 1}[y]
    )
)
```

### Item a)

Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
test_size = OJ.shape[0] - 800

X_train, X_test, y_train, y_test = train_test_split(
    OJ.drop(columns=["Purchase"]),
    OJ["Purchase"],
    test_size=test_size,
    random_state=299792458
)
```

### Item b)

Fit a support vector classifier to the training data using  $C = 0.01$ , with Purchase as the response and the other variables as predictors. How many support points are there?

```
model = SVC(kernel="linear", C=0.01).fit(X_train, y_train)

model.n_support_
```

array([309, 307])

Foram utilizados 616 suportes, sendo 309 do Citrus Hill e 307 do Minute Maid. O número de suportes foi bem próximo nas duas categorias.

### Item c)

What are the training and test error rates?

```
model.score(X_train, y_train)
```

0.76

```
model.score(X_test, y_test)
```

0.7074074074074074

Tanto os dados de treino quanto os dados de teste apresentaram acurácia de aproximadamente 70%, sendo os dados de treino apenas um pouco mais acurados que os de teste.

### Item d)

Use cross-validation to select an optimal  $C$ . Consider values in the range 0.01 to 10.

```
C_values = [0.01, 0.1, 0.5, 1, 2, 4, 6, 8, 10]
C_values

def cvval(mdl_search):
    return (
        pd.DataFrame(mdl_search.cv_results_)
        [["params", "mean_test_score"]]
        .sort_values('mean_test_score')
```

```

    )
mdl_search = {}

```

O objeto `mdl_search` é um dicionário para armazenar os modelos. O objeto `cval`, por sua vez, vai mostrar os parâmetros e a acurácia de cada modelo.

```

linear_svc = SVC(kernel = 'linear')

mdl_search.update(
    linear = skm.GridSearchCV(
        linear_svc, {'C': C_values},
        scoring='accuracy',
        cv=5
    )
)

mdl_search["linear"].fit(X_train, y_train)

cval(mdl_search["linear"])

```

	params	mean_test_score
0	{'C': 0.01}	0.69500
2	{'C': 0.5}	0.82625
7	{'C': 8}	0.82750
6	{'C': 6}	0.82750
8	{'C': 10}	0.82750
1	{'C': 0.1}	0.82875
3	{'C': 1}	0.82875
4	{'C': 2}	0.82875
5	{'C': 4}	0.83250

Os resultados mostram que o custo que maximiza a acurácia, no intervalo testado, deve estar próximo de 4. Apesar disso, a diferença entre os custos não foi muito grande, com  $C = 0,01$  sendo o único dos valores cuja acurácia ficou abaixo de 80%.

### Item f)

Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.

```

rbf_svc = SVC(kernel = 'rbf')

mdl_search.update(
    rbf = skm.GridSearchCV(
        rbf_svc, {'C': C_values},
        scoring='accuracy',
        cv=5
    )
)

```

```

)
mdl_search["rbf"].fit(X_train, y_train)
cval(mdl_search["rbf"])

```

	params	mean_test_score
0	{'C': 0.01}	0.61625
1	{'C': 0.1}	0.61625
2	{'C': 0.5}	0.61625
3	{'C': 1}	0.61625
4	{'C': 2}	0.61625
5	{'C': 4}	0.61625
6	{'C': 6}	0.61625
7	{'C': 8}	0.61625
8	{'C': 10}	0.61625

Para `gamma = 1`, todos os modelos apresentaram a mesma acurácia média, para qualquer que fosse o custo dentre os testados.

Uma vez que o custo tem baixa influência, variá-lo individualmente não é a estratégia mais adequada para teste.

Ainda seria possível, por exemplo, variar o valor de `gamma`, conforme o bloco a seguir.

```

mdl_search.update(
    rbf_diffgamma = skm.GridSearchCV(
        rbf_svc, {'C': C_values, "gamma": [3, 5]},
        scoring='accuracy',
        cv=5
    )
)

mdl_search["rbf_diffgamma"].fit(X_train, y_train)
cval(mdl_search["rbf_diffgamma"])

```

	params	mean_test_score
0	{'C': 0.01, 'gamma': 3}	0.61625
1	{'C': 0.01, 'gamma': 5}	0.61625
2	{'C': 0.1, 'gamma': 3}	0.61625
3	{'C': 0.1, 'gamma': 5}	0.61625
5	{'C': 0.5, 'gamma': 5}	0.67125
4	{'C': 0.5, 'gamma': 3}	0.68375
17	{'C': 10, 'gamma': 5}	0.70000
15	{'C': 8, 'gamma': 5}	0.70125
11	{'C': 4, 'gamma': 5}	0.70250
13	{'C': 6, 'gamma': 5}	0.70250
10	{'C': 4, 'gamma': 3}	0.70250

	params	mean_test_score
14	{'C': 8, 'gamma': 3}	0.70250
16	{'C': 10, 'gamma': 3}	0.70375
9	{'C': 2, 'gamma': 5}	0.70375
8	{'C': 2, 'gamma': 3}	0.70375
12	{'C': 6, 'gamma': 3}	0.70500
7	{'C': 1, 'gamma': 5}	0.70875
6	{'C': 1, 'gamma': 3}	0.71375

O melhor dentre os testes foi o modelo com `gamma = 3` e `C = 1`. Apesar disso, mesmo o melhor modelo dentre esses não atingiu a acurácia da maioria dos classificadores lineares, que foram consistentemente superiores ao kernel radial.

### Item g)

Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree = 2.

```
poly_svc = SVC(kernel = "poly", degree=2)

mdl_search.update(
    poly = skm.GridSearchCV(
        poly_svc, {'C': C_values},
        scoring='accuracy',
        cv=5
    )
)

mdl_search['poly'].fit(X_train, y_train)

cval(mdl_search['poly'])
```

	params	mean_test_score
0	{'C': 0.01}	0.61625
1	{'C': 0.1}	0.61625
2	{'C': 0.5}	0.61625
3	{'C': 1}	0.61625
4	{'C': 2}	0.61625
5	{'C': 4}	0.61625
6	{'C': 6}	0.61625
7	{'C': 8}	0.61625
8	{'C': 10}	0.61625

Selecionando 2 como o grau do polinômio, todos os modelos tiveram a mesma acurácia, que, por sua vez, foi igual à acurácia dos modelos de kernel radial com `gamma = 1`, de forma que o custo não teve influência.

Testando com outros graus de polinômio, obtemos:

```

mdl_search['poly_diffdegree'] = skm.GridSearchCV(
    poly_svc, {'C': C_values, 'degree': [3, 4, 5]},
    scoring='accuracy',
    cv=5
)

mdl_search['poly_diffdegree'].fit(X_train, y_train)

cval(mdl_search['poly_diffdegree'])

```

	params	mean_test_score
0	{'C': 0.01, 'degree': 3}	0.61625
24	{'C': 10, 'degree': 3}	0.61625
23	{'C': 8, 'degree': 5}	0.61625
22	{'C': 8, 'degree': 4}	0.61625
21	{'C': 8, 'degree': 3}	0.61625
20	{'C': 6, 'degree': 5}	0.61625
19	{'C': 6, 'degree': 4}	0.61625
18	{'C': 6, 'degree': 3}	0.61625
17	{'C': 4, 'degree': 5}	0.61625
16	{'C': 4, 'degree': 4}	0.61625
15	{'C': 4, 'degree': 3}	0.61625
14	{'C': 2, 'degree': 5}	0.61625
25	{'C': 10, 'degree': 4}	0.61625
13	{'C': 2, 'degree': 4}	0.61625
11	{'C': 1, 'degree': 5}	0.61625
10	{'C': 1, 'degree': 4}	0.61625
9	{'C': 1, 'degree': 3}	0.61625
8	{'C': 0.5, 'degree': 5}	0.61625
7	{'C': 0.5, 'degree': 4}	0.61625
6	{'C': 0.5, 'degree': 3}	0.61625
5	{'C': 0.1, 'degree': 5}	0.61625
4	{'C': 0.1, 'degree': 4}	0.61625
3	{'C': 0.1, 'degree': 3}	0.61625
2	{'C': 0.01, 'degree': 5}	0.61625
1	{'C': 0.01, 'degree': 4}	0.61625
12	{'C': 2, 'degree': 3}	0.61625
26	{'C': 10, 'degree': 5}	0.61625

Mesmo variando o grau dos polinômios juntamente com o custo, a acurácia permanece a mesma.

Dessa forma, esses modelos obtiveram consistentemente o pior desempenho entre os testados.

### Item h)

De modo geral, a classe de classificadores lineares obteve acurácia muito acima de todos os outros modelos, tendo quase todos os testados uma acurácia maior que 80%.

Os modelos polinomiais foram piores que todos os outros, com exceção dos modelos de kernel radial com `gamma = 1`. O desempenho destes foi pior até mesmo que o pior dos classificadores lineares.

Apesar de os modelos com o kernel radial terem ficado melhores para outros valores de  $\gamma$ , ainda há classificadores com a mesma acurácia dos anteriores de mesmo kernel. Todos esses possuem os valores mais baixos para o curso  $C$ .